INTERACTIVE COMPUTER PREPARATION OF SCHOOL TIMETABLES

A thesis presented for the degree of
Doctor of Philosophy in Electrical Engineering
in the University of Canterbury,
Christchurch, New Zealand

by

R.W. Platts B.Sc.(Hons)

1976

ABSTRACT

The preparation of school timetables is a task which requires 'human' decisions to be made yet which is tedious and time-consuming when carried out by hand. This thesis presents a new interactive timetabling system which shares the task between man and machine according to the respective abilities of each. The system was designed with two objectives in mind. The first was to enable the timetabler to observe the state of the timetable and to manipulate lessons as easily as he could in the manual method. This was achieved by the provision of computer-generated displays and flexible timetabling operations in the system. A strong relationship between display formatting and efficacy in interaction is revealed.

The second objective was to apply the power of the computer to situations which the manual timetabler finds 'difficult'. The computer techniques developed from theoretical methods in the literature include infeasibility tests to detect certain problem situations at early construction stages and tree searching methods for interchanging lessons. It is shown that the interactive application of the tree searching methods simplifies the solution of timetabling subproblems which require human decisions on changes to lesson constraints and distribution patterns.

The system was successfully used for the preparation of time-tables for a high school with a roll of about 1,000 pupils.

ACKNOWLEDGEMENTS

I am indebted to my supervisor, Dr J.H. Andreae, for introducing me to the school timetabling problem (on which he had worked for two years beforehand) and for his guidance and encouragement during the course of the project.

The enthusiasm and assistance of Mr T.R. Hitchings, Headmaster of Riccarton High School, and of Mr N.J. Gale and Mr K.L. Henderson, staff members in charge of timetabling, has been much appreciated. Their support and the co-operation of the remainder of the school staff have been vital to the success of the project.

The financial support of the University Grants Committee is gratefully acknowledged.

CONTENTS

CHAPTER 1

INTRODUCTION

The timetable is an important and essential document for the administration of the modern secondary school, both in New Zealand and elsewhere. It forms the means by which all teaching activity in the school is co-ordinated.

However, because educational curricula, staff and pupil rolls change from year to year, the timetable cannot remain static but must be changed accordingly. This nearly always means that a new timetable must be prepared from scratch at the beginning of each year. The responsibility of timetable preparation normally falls on either the headmaster or one or more senior staff members who may spend many hours of their valuable time on the timetable. The magnitude of the task is such that there are sometimes long delays before the timetable is ready for use. The school is sometimes forced to use a poor quality temporary timetable while the permanent one is being constructed. Although usually an effort is made to produce a good timetable, some schools may accept educationally undesirable arrangements of lessons just so that the timetable is ready on time.

Understandably, there has been considerable interest in the use of computers for constructing timetables. Many methods for computer timetabling have been devised, some of which are described in this thesis. However, computer-produced timetables have not always been satisfactory for all schools because the computer is not able to make the 'human' decisions which are an essential part of timetable construction procedure.

In this thesis, a new interactive method for school timetable preparation is presented. This method was developed from an approach proposed by Higgens and Andreae (1970) for sharing the task between man and computer in such a way that the tedious mechanical component of timetabling is undertaken by

human timetabler. This thesis shows that this approach is practical. It is emphasized that high quality timetables can be produced only if the human decision-maker is permitted to play a part during the construction process. School timetabling is not a problem to be solved by the computer alone, even though the timetable problem has a well defined basic mathematical structure. The 'human' aspect of practical school timetabling is far too important to be ignored or displaced out of the main timetable construction process.

The success of this interactive approach is demonstrated in the application of the system to the timetabling problem of Riccarton High School, Christchurch, New Zealand, which has a roll of about 1,000 pupils and is typical of a New Zealand state co-educational secondary school.

A condition that has been found to be important to the success of the method is that the timetabler must feel that he is making progress on his timetable at all times. Situations in which the timetabler spends a large amount of time with little benefit are liable to discourage him to the extent that he may give up using a computer aid. An essential prerequisite for continued progress in construction is that there must be fast, efficient communication between man and machine.

Direct man-machine communication has been made practical by the advent of on-line CRT terminals and graphic displays. These devices have opened up new applications for computers in fields such as management information, computer-aided design and on-line text and program editing. However for a problem with the characteristics of school timetabling, which involves a large quantity of information in which the detail is important, the design of the man-machine communication interface is not a simple straight-forward procedure. Design decisions must take into account the basis on which information is selected to be presented to the timetabler and the format in which it is to be presented on the display screen. These aspects influence the time required by the timetabler

that he is making progress if he is required to spend a large proportion of time extracting information from displayed data. The first contribution of this thesis is the presentation of a comprehensive system of display formats and interaction techniques that achieve this objective of fast man-machine communication.

The second general contribution made by this thesis is in dealing with the difficult problems that arise in timetabling when changes have to be made to an existing partially complete timetable in order to allow progress to be made. Such problems cannot be dealt with adequately by simple interactive techniques and require more sophistication on the side of the computer. Methods proposed for dealing with these problems include extensions to existing techniques for detecting 'impossible' situations at early stages of construction, as well as new algorithms for interchanging timetabled lessons to allow new lessons to be fitted. An interactive procedure is also proposed for dealing with difficult timetabling problems which require changes to the basic lesson structure.

Chapter 2 introduces the timetabling problem and describes the complexities that are characteristic of the practical problem. The underlying mathematical structure of the problem is formalised. The notation introduced is used in the remainder of the thesis. The standard manual technique of timetable construction is illustrated for comparison with the computer method.

Chapter 3 surveys school timetabling methods presented in the literature. Many of these methods are theoretically orientated and are applicable only to timetabling problems of very simple structure. It is argued that an effective practical timetabling system must be interactive.

Chapter 4 traces the development by the author of timetable display formats and interaction techniques from early unsuccessful designs to those found to be the most practical. Some general conclusions drawn from the experience gained in

theory of Newell and Simon (1972) to timetabling.

Attention is then turned towards the area of the second general contribution of this thesis. Chapter 5 examines the problem of determining whether or not a given partially completed timetable has a solution. A method proposed in the literature for testing the feasibility of 'mathematical' timetable problems is extended to make it applicable to more realistic problems. An infeasibility test suitable for use in an interactive environment is derived from this modified method. Another infeasibility test intended for use before timetable construction is started is based on a method for finding maximal complete subgraphs of a graph.

It is recognised that these tests cannot prevent all types of infeasibilities from occurring. Chapter 6 presents various algorithms for 'escaping' from infeasible situations by interchanging lessons already present in the timetable. These algorithms are fully applicable to the practical problem with all its complexities. Chapter 6 then continues by analysing the difficult infeasibility problems that require changing of the basic lesson structure for their solution. Since 'human' decisions are involved, the procedure suggested for solving such problems is interactive. Without such a solution procedure, problems of this type can be a source of considerable frustration for the timetabler.

Chapter 7 gives the results of application of the interactive computer aid to the timetable construction problems of Riccarton High School to illustrate the improvement over the manual method. Suggestions are then given for implementing the system to make it generally available to schools. Concluding remarks are presented in Chapter 8.

The material presented in this thesis was first described in the following publications:-

Andreae J.H., Gale N.J., Henderson K.D. and Platts R.W.; 'Developing an interactive aid for school timetabling'. Proceedings of the Third National Conference of the New Zealand Computer Society, Vol. 2, pp 182-202, August 1972.
— Manual method, initial proposal for current interactive system.

Platts R.W.; 'A school timetabling program'. In Man-Machine Studies, ed. J.H. Andreae, Rept UC-DSE/2, University of Canterbury, New Zealand, August 1973.
— Formal definition of timetable problem, infeasibility testing.

Platts R.W.; 'Computer-aided school timetabling'. In Man-Machine Studies, ed. J.H. Andreae, Rept UC-DSE/3, University of Canterbury, New Zealand, December 1973.
— Practical application to timetable construction in 1973.

Platts R.W.; 'An interactive approach to school timetabling'. In Man-Machine Studies, ed J.H. Andreae, Rept UC-DSE/4, University of Canterbury, New Zealand, September 1974.
— Techniques for interaction, tree searching methods, practical application in 1974.

CHAPTER 2

THE SCHOOL TIMETABLING PROBLEM

2.1 Introduction

The purpose of this introductory chapter is to attempt to define the timetable problem clearly. The term 'school timetable preparation' may refer to all activity carried out by the school directed towards producing a timetable at the end, or it may be restricted to the process of allocation of lessons to periods. The restricted sense will be considered first. After a verbal definition of the basic problem that must be solved in timetable construction, the complexities that characterise practical timetabling will be outlined. This is followed by a formal definition of the underlying mathematical problem. This definition also serves to introduce the terminology used in the remainder of the thesis.

The problem in its wider context is then considered and the manual procedure followed by Riccarton High School is used as an illustration of a typical method of solution. Other manual timetable construction methods are also briefly outlined to show the variety of methods that exist. Areas in which computer assistance would be desirable are identified, but specific problems in the wider context, such as course scheduling for individual students, are not developed any further.

2.2 The Basic Timetable Problem

For an introductory verbal definition of the timetable construction problem, we quote from Lions (1968):

"1) A school consists of teachers, students and rooms.
2) The student population may be divided into groups which are uniquely defined, and which enter into the timetable construction as individual units. Such a group is called a class.
3) The school week is divided into a certain number of periods (of time) of equal value.

4) The business of the school consists of a series of meetings. Each meeting involves a group of teachers, classes and rooms, and is assigned to a particular period or set of consecutive periods.
5) A timetable is a list of meetings (together with assigned periods) for members of a single school, which extends over a given period of time (usually one week).
6) The timetable problem is to construct a timetable for a given situation, which satisfies certain imposed conditions."

In the simplest form of timetabling problem, each meeting involves exactly one teacher and exactly one class, and occupies exactly one period. The only condition that must be satisfied in this problem is that teacher t_i must meet class c_j for a certain number r_{ij} of periods as stipulated by the school. The matrix $R = [r_{ij}]$ will be termed the simple requirements matrix to distinguish it from requirements matrices of different structure that will be introduced later. A 'requirement' is the statement stipulating that a certain teacher must meet a certain class for a given number of periods. Each element r_{ij} represents one requirement.

2.2.1 Example

To illustrate a timetabling problem of this simple form, suppose that a school consists of four teachers a,b,c,d and three classes A,B,C, and there are four periods in the week. Suppose that the simple requirements matrix to be satisfied by the timetable is as shown in figure 2.1.

		Teachers				
		a	b	c	d	Number of Periods Required
Classes	A	2	1	0	1	←
	B	1	1	1	1	
	C	1	2	1	0	

Figure 2.1 Simple Requirements Matrix

This matrix states that class A is to be taught by teacher a for 2 periods, by teacher b for 1 period, and so on. A timetable that satisfies this matrix is shown in figure 2.2.

		Periods 1	2	3	4
Classes	A	a	a	b	d
	B	b	b	c	a
	C	c	d	a	b

Teachers meeting classes

Figure 2.2 Timetable

In this timetable, class A is being taught by teacher a in periods 1 and 2, by teacher b in period 3, and by d in period 4. Also, in period 1, class B is being taught by teacher b, and class C by teacher c. Notice that no teacher is required to teach more than one class in any given period and that the number of periods for each class-teacher combination corresponds to the appropriate entry in the simple requirements matrix.

It may appear from this example that timetables can be constructed by a simple straight-forward procedure of inserting class-teacher meetings one-by-one into periods in which they do not conflict. However, in fact the use of this procedure is likely to result in a partially constructed timetable which cannot be completed (figure 2.3).

	a	b	c	d
A	2*	1*	0*	1*
B	1*	1*	1*	1*
C	1	2	1	0

Requirements

	1	2	3	4
A	a	a	b	d
B	b	d	a	c
C				

Timetable (a)

	a	b	c	d
A	2*	1*	0*	1*
B	1*	1*	1*	1*
C	1*	2	1	0

Requirements

	1	2	3	4
A	a	a	b	d
B	b	d	a	c
C				a

Timetable (b)

* satisfied requirements

Figure 2.3 Timetable Problem Which Cannot

In figure 2.3(a) the requirement that teacher a meets class C for one period is still to be satisfied. Period 4 is the only period available in which an allocation of a to C can be made, since a is already teaching in periods 1,2, and 3 (figure 2.3(b)). After allocating a to C in period 4, there are two periods required for meetings between b and C. One of the meetings can be in period 2. However it is not possible to allocate the second. In periods 1 and 3 b is teaching other classes, and in period 4 class C is being taught by teacher a.

Although it is possible to strike infeasible situations such as this one in the course of construction, a theorem due to Konig (1950) can be applied to show that problems of this simple type always have a solution, provided that the requirements matrix satisfies the following condition.

Condition 2.1 $\sum_i r_{ij} \leqslant$ number of periods in the week

and $\sum_j r_{ij} \leqslant$ number of periods in the week.

Csima (1965) describes a solution method based on this theorem. The timetable is constructed period by period. For each period, non-zero elements in the simple requirements matrix are chosen and class teacher allocations are made to the period until no further can be made. Each requirements matrix element chosen is reduced by 1. Allocations in each period must be made in such a way that the row and column sums of the resulting matrix are not greater than the number of periods remaining. In other words condition 2.1 must be satisfied by the timetable problem remaining. This is ensured by including in the choice of allocations those from rows and columns whose sums are initially equal to the number of remaining periods. If this procedure is followed an infeasible situation will never occur, provided of course that condition 2.1 was satisfied at the start.

2.3 Complexities of the Practical Problem

Theoretical models can be constructed for many practical problems but in most cases the model is only an approximation to the real problem. This is true for school timetabling. However the complexities that characterise practical school timetabling can be identified and defined relatively easily:

1) Blocks. Included in the school's requirements may be the request that a teacher t_{i1} meets a class C_{j1} at the same time as another teacher t_{i2} meets another class C_{i2} and so on, at the same time as t_{in} meets C_{in}. Alternatively it may be required that classes C_{j1} to C_{jn} combine together to form a large group G which is then divided into smaller groups or sets g_1, $g_2 \ldots g_m$ which meet the teachers t_{i1}, $t_{i2} \ldots t_{im}$ respectively. Both of these types of requirements imply that a meeting may consist of more than one class. Such a meeting is termed a block. School situations for which blocks are required include the following:

   a) A selection of subjects is offered as options to the pupils in a set of classes. Thus g_k is the set of all pupils in the classes C_{j1} to C_{jn} who have chosen subject s_k, which is taught by teacher t_{ik}. This is the most common reason for blocking. In some schools option blocks may occupy a large proportion of teaching time to the extent that the classes C_1, $C_2 \ldots$ exist as administrative units only.

   b) The pupils in the classes involved are streamed for a particular subject according to ability. This reason for blocking is nowadays declining in importance.

   c) Practical classes such as woodwork and metalwork may be blocked together to make better use of the available facilities.

d) The classes involved may alternate or rotate amongst several subjects over the course of the year.

e) It may be necessary to have smaller groups for some subjects because of limited specialist room sizes for example.

f) For some lessons the classes must be segregated according to sex.

2) Double and Triple Periods. For subjects such as Art and Woodwork a single period is too short for effective teaching. Two or three consecutive periods may be required and these periods should not straddle a morning or lunch break. The requirements must therefore state the number of double and triple periods required as well as the number of single periods.

3) Classrooms. Each meeting in the timetable will require classrooms, one for each teacher involved in the meeting. (A few exceptions may occur for some subjects, for example, Physical Education). Subjects such as woodwork and science require specialist rooms, which may be limited in number. For other subjects 'ordinary' classrooms are sufficient. There is sufficient flexibility in the choice of these 'ordinary' classrooms to enable their allocation after the main timetable construction has been completed. During the main construction the only restriction that must be checked is that the number of classrooms required does not exceed the number of rooms available in the school in any period.

4) Period Restrictions. Some meetings must occur at fixed times, for example, to coincide with radio broadcasts. Others must not occur at certain times, because one or more of the teachers may not be available. For example, some teachers have to attend university lectures and senior staff may have external commitments.

5) Distribution. Lessons in a particular subject for a given class must be well spread through the week. For example if 5 periods of English are required for class 3A, then they must not all be allocated to Monday. The optimum distribution of one English period per day is preferred but two periods on one day and none on another will be accepted if the optimum cannot be achieved. As important as subject distribution is teaching and non-teaching period distribution for each teacher. Here a variety of conditions apply, such as 'at least one non-teaching period a day if it is possible' or 'teaching periods for part-time teachers must be well grouped together'.

2.4 Formalisation of the Practical Timetabling Problem

In section 2.2 a very simple model of the timetable problem was presented. A model which comes closer to the practical problem with its complexities will now be introduced. The terminology introduced in this model will be used in the remainder of this thesis. The model formalises blocks and classrooms, but not double periods, distribution constraints or period restrictions. The inclusion of these 'period-dependent' constraints would unjustifiably increase the complexity of the model.

2.4.1 Items

Instead of regarding teachers and classes as distinct and independent sets, we now adopt the unifying concept of 'items' introduced by Johnston and Wolfenden (1968):-

> "An item is defined as (a) a class, or (b) a teacher, or (c) a type of classroom, or (d) a piece of special equipment which does not form part of the equipment of a particular classroom.
>
> An item may have more than one life if it is capable of doing several jobs at the same time.
>
> Items of type (a) and (b) have only one life, but under type (c) we might have three identical science laboratories

which are considered as a single item with 3 lives, while under type (d) we may have two portable tape-recorders which form one item with two lives".

The symbol L_k will be used to represent the total number of lives associated with item k in the school. Each meeting in the timetable therefore involves a collection of one or more lives of several items. The number of lives of any item k in the meeting must of course be less than L_k.

2.4.2 Requirements

The presence of blocks in practical problems (section 2.3) implies that the concept of requirement must be extended to allow for more than one class and/or more than one teacher. The term 'requirement' is now defined as the statement stipulating that a collection of one or more lives of several items must meet simultaneously for a given number of periods. The simple requirements matrix $R = [r_{ij}]$ must be replaced by two new matrices, the requirement-item matrix $X = [X_{rk}]$ and the requirement-period vector $N = [N_r]$. The allocation of items to meetings is determined by the set of requirements which form part of the initial data supplied by the school. These requirements are numbered from 1 to r_{max}. Information for requirement r is given by a row of the requirement-item matrix $X = [X_{rk}]$ defined below, and by an element of the requirement-period vector $N = [N_r]$, also defined below. Both X and N have positive integral elements. Each meeting in the timetable is associated with exactly one requirement.

Definition 2.2. The requirement item matrix $X = [X_{rk}]$ is defined by X_{rk} = the number of lives of item k required by requirement r

= the number of lives of item k involved in any meeting associated with requirement r.

Definition 2.3. The requirement-period vector $N = [N_r]$ is defined by N_r = the number of periods required for meetings associated with requirement r.

2.4.2.1 Example

Suppose that the set of items in a hypothetical school consists of:

a) four classes a,b,c,d
b) five teachers p,q,r,s,t
c) one classroom type x with 2 lives

and suppose that there are 3 periods in the school week.

An example of requirement information appropriate to this school is given in figure 2.4.

Matrix X:

	classes				teachers					specialist classrooms
	a	b	c	d	p	q	r	s	t	x
A	1	1	0	0	1	0	0	0	1	2
B	0	0	1	1	0	1	0	1	0	0
C	1	0	0	0	0	1	0	0	0	1
D	0	1	0	0	1	0	0	0	0	0
E	0	0	1	0	0	0	1	0	0	0
F	0	0	0	1	0	0	0	0	1	1
G	1	0	0	0	0	0	0	0	1	0
H	0	1	1	0	0	0	1	1	0	2
I	0	0	0	1	1	0	0	0	0	0
L_k	1	1	1	1	1	1	1	1	1	2

Vector N:

A	1
B	1
C	1
D	1
E	1
F	1
G	1
H	1
I	1

Figure 2.4 Example of Requirement-Item Matrix & of Requirement-Period Vector

Each requirement has been given a capital letter. Requirement A, for example, states that classes a and b meet teachers p and t for one period of the week. Two rooms of type x are also needed. Note that there is no indication of whether a meets p and b meets t, or a meets t and b meets p. This information is irrelevant to the timetable problem as a block is considered as a single entity.

2.4.3 Clashes

Definition 2.4. Requirement r_1 clashes with requirement r_2 (written $r_1 * r_2$) if there exists an item k such that

$$X_{r_1 k} + X_{r_2 k} > L_k.$$

If requirement r_1 clashes with requirement r_2, then a meeting associated with r_1 cannot be entered into the same period as a meeting associated with r_2. This is because the total number of lives required for item k exceeds the number available. In figure 2.4, requirement B clashes with requirement E because:

$$X_{Bc} + X_{Ec} = 2 > L_c = 1.$$

Both requirements involve class c, but there is only one life of this item in the school.

We can extract all clash information from the X matrix and summarise it in the form of a clash matrix or an equivalent clash graph:

Definition 2.5. The clash matrix $C = \left[C_{r_1 r_2} \right]$ is defined by

$$C_{r_1 r_2} = 1 \quad \text{if} \quad r_1 * r_2$$

$$= 0 \text{ otherwise.} \quad \text{(This matrix is symmetric)}$$

Definition 2.6. The clash graph G = (V, E) is the graph with vertex set $V = \{v_1, v_2, \ldots, v_{r_{max}}\}$ and edge set

$E = \{ (v_{r_1}, v_{r_2}) : r_1 * r_2 \}$. (See Harary (1969) or Berge (1962) for basic definitions in graph theory).

Figure 2.5 shows the clash matrix and clash graph defined from the arrays of figure 2.4.

	A	B	C	D	E	F	G	H	I
A	1	0	1	1	0	1	1	1	1
B	0	1	1	0	1	1	0	1	1
C	1	1	1	0	0	0	1	1	0
D	1	0	0	1	0	0	0	1	1
E	0	1	0	0	1	0	0	1	0
F	1	1	0	0	0	1	1	1	1
G	1	0	1	0	0	1	1	0	0
H	1	1	1	1	1	1	0	1	0
I	1	1	0	1	0	1	0	0	1

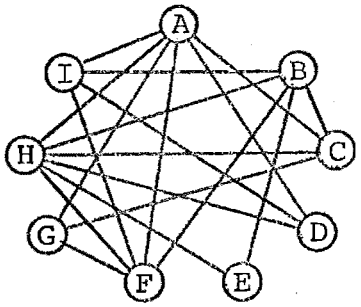


Figure 2.5 Example of Clash Matrix & Clash Graph

In Chapter 5 it will be shown that 'maximal mutually clashing sets' are important in practical timetabling for showing infeasibility. The maximal mutually clashing set can be defined in terms of the previously introduced terminology.

Definition 2.7. A mutually clashing set (MCS) is a subset S of $\{1, 2, 3, \ldots, r_{max}\}$ whose elements satisfy the property $r_1 * r_2$ for all $r_1, r_2 \in S$.

Definition 2.8. A maximal mutually clashing set (MMCS) is a MCS which is not a proper subset of any other MCS.

A MCS is represented by a complete subgraph (Harary, 1969) in the clash graph G. A MMCS is represented by a maximal complete subgraph (clique) in G.

An example of a MMCS is the set {B, F, H} of the requirements of figure 2.4. This set is a MMCS since

B and F clash because item d is common to both
F and H clash because of item x
H and B clash because of items c and s,

and it is not a proper subset of any other MCS that can be formed from the requirements of figure 2.4.

2.4.4 The Timetable

A definition of the timetable itself will now be given in terms of the conditions it must satisfy.

Definition 2.9. A timetable (complete timetable) which satisfies the requirements defined by the arrays X and N, is a zero-one matrix (a matrix whose elements are zeros and ones) $T = \left[T_{rp} \right]$ satisfying the conditions:

a) $\forall r, \sum_{p} T_{rp} = N_r$

b) $\forall k, p, \sum_{r} X_{rk} T_{rp} \leqslant L_k$

where r is an index over requirements
p is an index over periods.

A '1' in the matrix element position T_{rp} indicates that requirement r has been assigned to period p. The allocation of a requirement to a period will be termed an 'assignment'.

In definition 2.9, condition a) states that the number of assignments in the timetable for the requirement r must be equal to the number specified by the school, i.e. N_r. Condition b) is the no-clash condition which states that the total number of lives of any item k required in any period must not exceed the number L_k available in the school.

From condition (b),

$$\sum_p \sum_r X_{rk} T_{rp} \leqslant \sum_p L_k$$

which implies

$$\sum_r X_{rk} \sum_p T_{rp} \leqslant p_{max} L_k$$

where p_{max} is the number of periods in the week. Using condition (a), we obtain:

Condition 2.10.

$$\sum_r N_r X_{rk} \leqslant p_{max} L_k$$

A necessary condition for the timetable to exist is that the set of requirements satisfy condition 2.10. For teachers and classes, which are items with $L_k = 1$, this simplifies to:

Condition 2.10'.

$$\sum_r N_r X_{rk} \leqslant p_{max}$$

This states that the total required number of teaching or lesson periods must be less than the total number of periods in the week. The condition is equivalent to condition 2.1 which applies to the basic timetabling problem of section 2.2.

If $L_k = 1$ for all items k, then the clash matrix C or the clash graph G can be used in place of the X matrix as a source of information for timetable construction. Furthermore, by defining a new clash graph G' in which each vertex v_r of the old clash graph G is replaced by a complete subgraph of N_r vertices, it is possible to represent the information in both the R and X arrays in graphical form. The solution to the timetabling problem is then a p_{max}-colouring of the graph G' in which no two adjacent vertices have the same colour. This representation is used in at least one proposed method of timetable construction (section 3.2.5).

If the condition $L_k = 1$ does not apply to all items in the school, then the clash matrix or graph does not give sufficient information for timetabling.

For example if $L_{k_o} = 2$ for some item k_o and $X_{r_1k_o} = X_{r_2k_o} = X_{r_3k_o} = 1$, then the pairs (r_1,r_2), (r_2,r_3), (r_3,r_1) do not clash, but r_1, r_2 and r_3 cannot be all entered together into one period of the timetable.

Definition 2.11. A partial timetable is a zero-one matrix $T' = [T'_{rp}]$ satisfying the conditions

a) $\forall r, \quad \sum_p T'_{rp} \leqslant N_r$

b) $\forall k, p, \quad \sum_r X_{rk}T'_{rp} \leqslant L_k.$

Definition 2.12. A partial timetable T' is termed feasible iff there exists a complete timetable T which contains T', i.e.

$\forall r, p, \quad T'_{rp} \leqslant T_{rp}.$

Definition 2.13. In a partial timetable T', the period p is free for the requirement r iff

$\forall k, \quad X_{rk} + \sum_j X_{jk}T'_{jp} \leqslant L_k,$

i.e. if the requirement r clashes with no assignments already present in period p.

2.4.4.1 Example of Timetable

A timetable which satisfies the requirement data of figure 2.4 is shown in figure 2.6.

		periods p			
	T_{rp}	1	2	3	N_r
	A	1	0	0	1
	B	1	0	0	1
	C	0	1	0	1
	D	0	1	0	1
requirements r	E	0	1	0	1
	F	0	1	0	1
	G	0	0	1	1
	H	0	0	1	1
	I	0	0	1	1

Figure 2.6 Example of Timetable T_{rp}

A more concise and informative representation of the timetable will be used later in this thesis. In this representation, the timetable is shown as a class-period matrix. An assignment has the format shown in figure 2.7, in which R is the requirement which involves the classes c_1, c_2, c_3 and which has been assigned to period p.

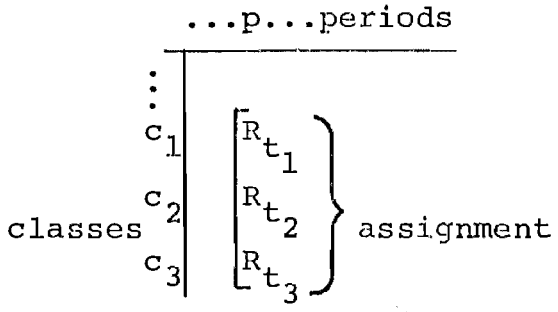


Figure 2.7 Representation of Assignment in Class-Period Format

The teachers involved in the requirement are t_1, t_2, t_3. There can be no ambiguities since each class has a life number of 1 and therefore not more than one requirement can involve any given class in any given period. Classroom types and other multiple-life items are represented in additional matrix rows. For each classroom type k, L_k rows are added. The label of a requirement r involving classroom type k appears in X_{rk} of the rows.

The timetable of figure 2.6 is shown in class-period format in figure 2.8. Only blocks are bracketed.

		1	2	3 periods
classes	a	A_p	C_q	G_t
	b	A_t	D_p	H_r
	c	B_q	E_r	H_s
	d	B_s	F_t	I_p
room type x	x	A	C	H
	x	A	F	H

Figure 2.8 Timetable in Class-Period Format

A partial timetable which is not feasible is shown in figure 2.9.

	1	2	3
A	1	0	0
B	0	0	1
C	0	0	0
D	0	0	1
E	1	0	0
F	0	0	0
G	0	0	1
H	0	1	0
I	0	0	0

(a) T_{rp}

	1	2	3
a	A_p		G_t
b	A_t	H_r	D_p
c	E_r	H_s	B_q
d			B_s
x	A	H	
x	A	H	

(b) Class-period representation

Figure 2.9. Infeasible Partial Timetable

Period 2 is free for requirement I, since the class d is free and teacher p is not involved in any other assignment in that period. However there is no free period for requirement C.

2.5 Timetable Construction — The Manual Method

The overall process of timetabling is a long multistage one, which may begin 6 to 12 months before the timetable is required for use. To illustrate the wider context of timetable construction, the procedure followed by Riccarton High School will now be described.

2.5.1 Preparation of the Requirements

All of the pre-construction activity carried out by the school is concerned with the preparation of a plan of requirements (termed the 'block timetable'). The steps involved in this activity are shown in the flow diagram of figure 2.10. This diagram is adapted from that given by Scott (1969). In the following discussion numerals in brackets refer to the labelled steps in the diagram.

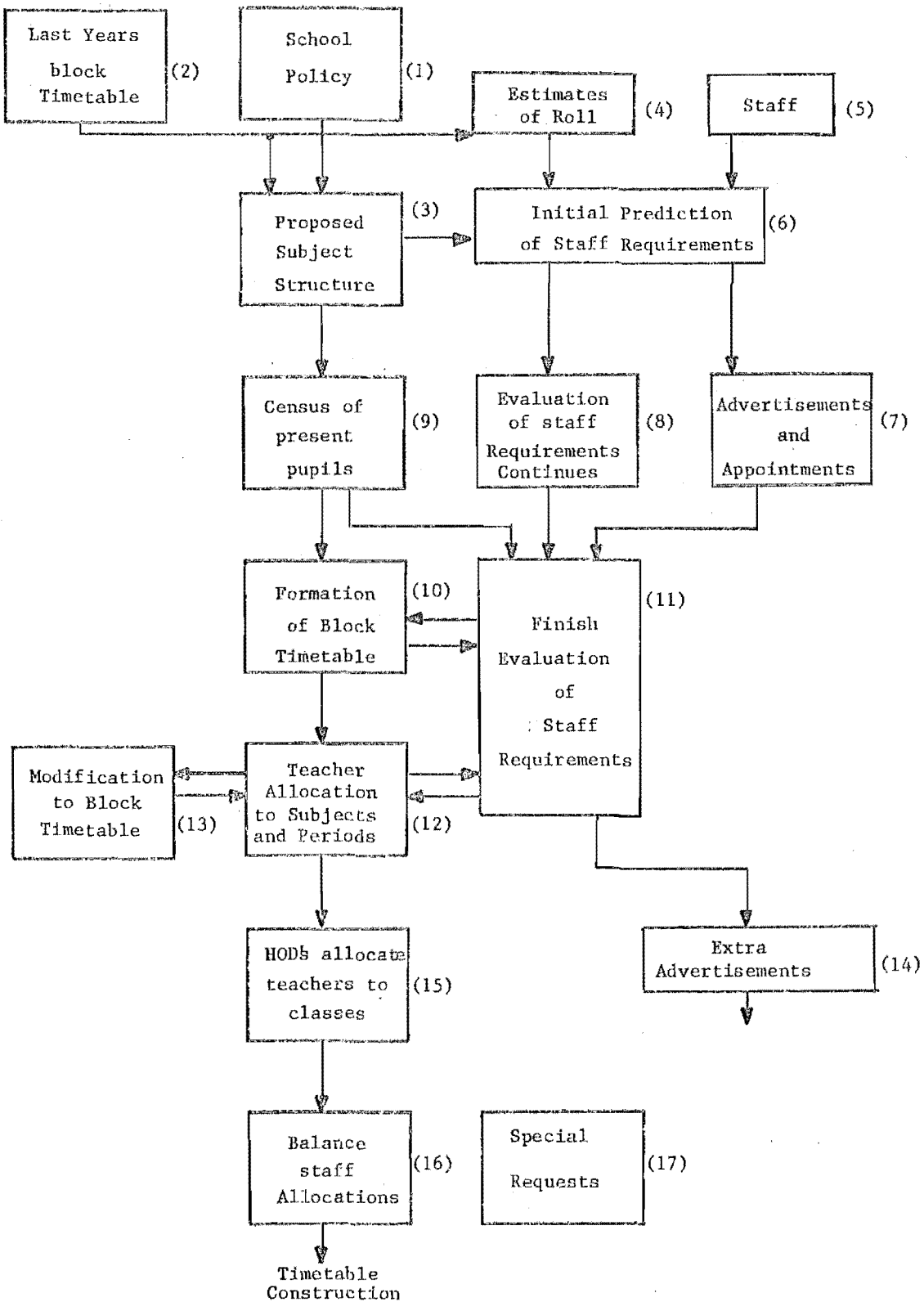


Figure 2.10 Sequence of Steps in Block Timetable Preparation

The School policy (1) formulated by the headmaster, comprises general educational aims and curriculum decisions, such as whether or not certain subjects will be offered at certain levels. Form this, and from the block timetable compiled the previous year (2), a proposed subject structure is drawn up. This subject structure gives in detail the choice of subjects to be offered to each form, for example, 5th form pupils must choose: English; one of Maths, Biology or General Science; one of History, Geography or Technical Drawing; and so on.

Also at a fairly early stage an estimate is made of the next year's roll (4) in each form, by projecting from the current year's rolls. (For some forms the roll is fixed). From this, and from the proposed subject structure approximations for the number of teacher-periods for each subject are estimated for the initial prediction of staff requirements (6). These teacher-period figures are compared with the number of present staff likely to be available in the following year (from (5)). The result is the approximate number of new staff that need to be employed for the following year. The headmaster advertises for the new staff (7) and appoints the minimum number that he expects he will need. Appointments take place about the middle of October. Meanwhile the evaluation of staff requirements continues (8) and the requirements gradually become more definite as more information becomes available.

Near the end of October, a census of present fourth, fifth and sixth form pupils is undertaken. Each pupil is asked whether or not he intends to return in the following year, and what subjects he would choose from the subject structure if he does return. From the information returned, a reasonably accurate estimate of the number of pupils taking each subject in each form can be made. These figures are used both for the formation of the block timetable (10) and for the final evaluation of staff requirements (11). In the formation of the block timetable, which takes place

about the beginning of November, pupils are allocated to classes according to their chosen courses. Lessons are blocked together where necessary. The aim is to obtain reasonable numbers in the groups for every lesson.

Soon afterwards the numbers of periods for each subject are totalled and the periods are distributed between members of the staff that are capable of teaching that subject (12). This may show up discrepancies between the expected numbers of staff available and the staff actually required. These discrepancies are eliminated either by altering the staff requirements or by changing the block timetable (13). For example, if too many teachers are required for some subject, either the staff requirements may be increased and extra advertisements issued (14) or classes for the subject may be increased in size and the number of periods for the subject decreased in the block timetable.

The actual allocation of the teachers to the lessons in the block timetable is done by the heads of the appropriate departments (15). Each Head of Department (H.O.D.) is given the number of periods allocated to each teacher in the subject or subjects of his department. The H.O.D. may make minor changes in the numbers, but must ensure that the total is not changed.

Finally, the teacher allocations are displayed on a large board in a matrix format of class versus teacher. The total number of teaching periods for each teacher is determined. The totals are balanced by transferring periods from one teacher to another, to obtain a fair distribution of the teaching load, taking into account the extra responsibilities that some of the staff may have. Also balanced are the ratio of men to women teachers and the ratio of senior to junior teachers for each class.

At about the same time, the staff present their requests for special conditions, such as periods free, to be satisfied in the timetable. These, together with the completed block timetable, are received by the timetabler at about the end of November. The second main stage of the overall process then begins - the construction of the timetable itself.

2.5.2 Construction of the Timetable

The block timetable (figure 2.11) gives the following information for each class:

1) The subjects taught.
2) The teachers taking the subjects (represented by two or three letter abbreviations).
3) The number of periods (single and/or double) for which the subject is to be taught (in the left hand column unless otherwise specified).

   Subjects which are blocked together are enclosed in rectangles.

Because of the size of the problem (35 periods by approximately 30 classes for Riccarton High School), a specially constructed board is generally used for allocating lessons to periods. The board has a matrix of nails or holes, and coloured tickets, pegs or other small objects which can be attached to the board are used to represent lesson-periods (figure 2.12). The nails or holes are usually arranged so that periods lie along the horizontal axis and either classes or teachers down the vertical axis. Variations of this may occur, for example, all the nails or holes of a period may be arranged in a rectangular block rather than in a column (Scott, 1969). Colour coding schemes are used to assist in timetabling. In the case of Riccarton High School, a class-period format is used and the teacher of a lesson is identified by the combination of a colour and a bold marking on the ticket. The subject, class(es) and the teacher's abbreviation are also written on the ticket.

The first task in timetable construction is to prepare tickets to represent the lessons in the block timetable. A ticket is prepared for each period required for each subject and each class. Two tickets are prepared for each double period. A block is represented by a set of tickets, one ticket for each teacher involved in the block. The task of ticket preparation is carried out by pupils and takes three or four of them about 5 or 6 hours.

Class →	3A	3B	3C	3D	3E	...
5	Eng McG	Eng Ws	Eng Bu	Eng Ca	Eng Wn	...
5	SS Bn	SS Tn	SS Wi	SS Eg	SS Dn	
5	Sci Hs	Sci Br	Sci Cs	Sci CS	Sci Br	
5	Maths McN	Maths Pr	Maths Bk	Maths Cr	Maths ED	
4	Fr Bu	Fr Ea	4 Fr Ea	C.St.Sk	TD An	Typ Wn
4	Ger Ca	Ger In	2d MW Mo	WW Hn	H.Ec. Fl	Clo Sp
2	Art Lo	Art Be	Art Lo	Art Be	Art Be	
2	Mus Gl	Mus Gl	Mus Gl	Mus Gl	Mus Gl	
3	PE Bt	PE Wa	PE Ly	PE Bt	PE McN	...
Number of periods						

Teacher

Subject

Figure 2.11 Portion of Block Timetable

Class-Period Format

	MONDAY						
	1	2	3	4	5	6	7
3A	.	Bn SS	.	.	.	.	.
3B	.	.	.	.	.	.	.
3C	.	.	.	.	.	.	.
3D	.	.	.	.	.	.	.
⋮							

Teacher-Period Format

	MONDAY						
	1	2	3	4	5	6	7
An	.	.	.	.	.	.	.
Be	.	.	.	.	.	.	.
Bk	.	.	.	.	.	.	.
Bn	.	3A SS	.	.	.	.	.
Br	.	.	.	.	.	.	.
⋮							

Figure 2.12 Timetabling Board

Configurations

The timetabler begins construction by allocating the largest blocks. These initial blocks are arranged to satisfy two main distribution requirements:

1) distribution of the subjects within the blocks over the week, and
2) uniformity of the number of blocks per day for each class.

The arrangement is important since it affects the distributions of later entries.

As the number of blocks in the timetable increases the choice of periods available for allocation of each entry decreases rapidly. Eventually blocks already in the timetable will have to be moved in order to free periods for new entries. Attempting to find a suitable non-clashing rearrangement of blocks can be a difficult and time-consuming process.

The difficulty involved in entering a block at this stage depends strongly on the arrangement of the blocks already in the timetable. If it appears impossible to complete the block allocations then the timetabler must do one of two things:

a) Remove all the blocks and start again, in the hope that a better arrangement may be obtained the second time, or
b) Assign another teacher to take a subject in a block, so that the block can be entered into a period in which it would otherwise clash.

In an attempt to reduce this difficulty, the timetabler may rely on various intuitive heuristic methods, such as to try to 'line blocks up' in periods of the timetable.

At every step of the block-entering process, the timetabler must determine whether or not a block can be entered into a period of the timetable and if not, what other blocks it clashes with.

This requires a visual scan of the tickets in the period to find if there are any teachers elsewhere which are in common with the ones involved in the block. This elementary step is a time-consuming and error-prone process. Typically the time required to search for just one teacher in a period of 40 tickets is about 9-15 seconds. This time is likely to increase with the number of teachers being searched for (although Neisser (1966) reports that, after much practice, subjects can find multiple targets as quickly as single targets). Furthermore, frequently the target ticket is missed and clashes may remain undetected for many moves. When all the large blocks have been entered, it is necessary to stop and carry out a complete overall clash check by marking off teachers on a separate piece of paper.

After fitting the larger blocks the timetabler continues by entering the smaller blocks (each consisting of about 2 teachers and 2 classes) and double-period lessons, and finally the unblocked single-period lessons. During the fitting of the smaller blocks problems can arise that are similar to those encountered during the large block stage. Considerable rearrangement may be necessary.

By contrast the initial stages of entering the singles are easy and rapid progress is made. The procedure followed is to fill the periods for each class in turn, proceeding from classes with a large number of blocks to those with fewer. Blocks are not moved at all at this stage unless it is found impossible to complete the lessons for a class.

Complete clash-checks must again be made, one when the timetable is two-thirds filled, and another when it is 95% filled.

The final stage of entering the last 5% of the lessons can be as difficult as the large block stage. The periods of the teachers and classes are nearly full. To get one lesson in may require a long series of moves of other lessons already in the timetable.

As in the earlier stages teachers may have to be reallocated for some of the lessons.

When the timetable is complete, a final exhaustive clash-check must be made. This can take about 3 to 4 hours. Overall, to complete the timetable on the board, two senior teachers at Riccarton High School require about 10 days at 6 hours a day - a total of 120 man-hours (Henderson, 1974).

However the timetable is not ready for use yet. The next task is to copy the timetable onto paper. A further 8 hours is needed to produce both a teacher-period and a class-period timetable. At the same time, the head of the science department makes a copy of the science timetable, which he uses to allocate the science laboratories.

The science laboratory allocations are then copied on to the master timetable, and rooms are assigned to the remaining lessons. This takes a further 3-4 hours. The timetable is then ready for use by the school.

The times for the various stages are summarized in table 2.13. The delay from when the block timetable is ready to when the final timetable with rooms is ready is about 2½ to 3 weeks for Riccarton High School.

1.	Preparation of Tickets	20 man-hours
2.	Timetable Construction	120 man-hours
3.	Copying Onto Paper	16 man-hours
4.	Rooms Allocated	4 man-hours
	Total:	160 man-hours

Table 2.13 Times For Manual Timetable Construction

2.6 Other Manual Timetabling Methods

Some schools are able to use a simplified subject structure wich reduces the magnitude of the task of timetabling to the extent that it is unnecessary to use a board. In this structure the subjects for each of the upper forms (5th, 6th and 7th) are blocked together into 5 or 6 blocks for each form. Each of these blocks has an equal number of periods. Before timetabling is started, the blocks are aligned into 5 or 6 columns to find an arrangement in which there are no clashes. Changes are made to the teacher allocation if necessary. The blocks in the 3rd and 4th forms are also aligned with these columns.

The first step in period allocation is simply to assign the columns of the block arrangement to the periods in a manner which gives good distribution. Each column is assigned to 6 or 7 periods. Then all that remains are singles which are allocated to the periods in the same way as described in section 2.5.2.

A similar method is the use of layouts (Lewis, 1961, Lawrie, 1969) Here the subjects for all classes in each form are blocked together before teachers are allocated. A layout is the set of blocks for a form. Blocks are fitted together in the timetable in such a way that the total number of staff required for each subject in any period does not exceed the number of suitable staff available in the school. The layout method is best suited to a school which has options in every form with the consequent high proportion of lessons involving sets of pupils drawn from several classes. However, for schools in which the classes remain intact for most of the lessons, the formation of layouts is a burden equivalent to timetabling and it tends to restrict flexibility unnecessarily.

2.7 Timetable Preparation by Computer

It can be seen from the foregoing that the process of preparing a timetable involves a great deal of time and effort on the part of senior staff every year. The computer promises to be a powerful tool which could relieve senior staff of many man-hours of

this effort. The greatest benefits are likely to be realised by applying the computer to the timetable construction stage, since:

a) this stage is especially tedious and difficult. Successful manual construction requires special skills and years of experience.

b) the stage is best suited for computer application. Timetable construction is a relatively mechanical process which involves a large quantity of data. The computer is well known for its ability to handle large volumes of data quickly and without error. The computer can also carry out basic functions such as clash checking and printing out far more quickly and efficiently than the human.

There has understandably been much interest in timetable construction by computer. Methods proposed in the past will be surveyed in the next chapter. However a full timetable preparation system must encompass more than just timetable construction. In particular, the following areas would benefit from computer assistance:

a) Processing of the pupil's census returns.
b) Grouping pupils together into classes.
c) Forming blocks.
d) Assessing teacher requirements for each subject.
e) Adjusting total teaching periods for each teacher.

Such a system would eliminate the conversion of the block timetable to input data for the computer, itself a tedious process wich is necessary for any type of construction-only system. Thus the implementation of a timetable construction system must be considered as only a first step towards a fully comprehensive timetabling system. Considerable extra design and programming effort will be needed in an area outside the scope of this thesis.

COMPUTER METHODS FOR TIMETABLING

3.1 The Area of Interest

This chapter surveys some of the methods that have been proposed for the computer solution of school timetabling and associated problems. School timetabling belongs to the general class of scheduling problems in which 'events' 'tasks' or 'jobs' requiring resources are to be allocated to times in such a way that the total amount of resource of any given type required never exceeds the amount of that type of resource available. Operations research problems such as job-shop scheduling fall into this class, but unlike school timetabling such problems have extra constraints in the form of ordering relations, for example, job A must be completed before job B is started. These ordering relations may be more restrictive than the resource availability constraints to the extent that the whole character of the problem is changed and different solution techniques are required.

It is therefore necessary to further restrict the area of interest in order to define a class of problems whose solution techniques are applicable to school timetabling. Such problems will have the following characteristics:

a) The time available for scheduling is divided into a fixed number of discrete periods.

b) Each 'event' occupies a small integral number of consecutive periods. A large proportion of the events occupy only one period.

c) Resources are discrete and most of the resource types consist of only one object (or 'life'). This implies that two events each requiring a resource of this kind can never be allocated to the same time period together. It is therefore possible to define simple 'clash' relations between the events.

d) The resources available constitute a finite set. Most of the resources are utilised for over 75% of the available time and some for 100% of the time.

e) The problem is chiefly one of determining the existence of a solution. However this does not exclude optimisation problems since one type of problem can usually be converted to the other. For example a solution exists in n periods if the minimum number of periods found using a time-optimising algorithm is n. Conversely the minimum time can be found by testing for existence in 1 period, 2 periods, 3 periods, and so on, until a solution is found.

Other timetabling problems which satisfy many of the above conditions are university and examination timetabling. The university timetabling problem is similar to that of school timetabling, except that the resources are not as heavily utilised. There is more 'slack' and 'human' conditions play a greater part in defining acceptable solutions. Thus although methods proposed for university timetabling problems are applicable to school timetabling, they will not be very effective as the basic algorithm is relatively straight-forward. The structure of examination timetabling problems differs slightly from that of the school counterpart, but the methods are still transferable. Similar considerations apply to examination timetabling as to university timetabling.

3.2 Computer Timetabling Methods

The methods surveyed in this chapter will be classified under headings according to the general approach taken. The headings are:

a) Heuristic Timetabling Methods
b) The Theoretical Approach of Gotlieb

c) Integer Programming
d) Tree Searching
e) Other Methods.

3.2.1 Heuristic Timetabling Methods

A heuristic method is one in which decisions are made by the application of rules derived from intuition. In a heuristic timetabling method, these rules determine which requirement to choose next and into what period the chosen requirement should be entered. Solutions are not guaranteed, but the heuristic rules are designed so that the method produces within a reasonable time a result that is close to a complete solution.

Appleby et al (1961) report a heuristic method that represents one of the earliest attempts at timetabling by computer. The basic philosophy behind the method is to choose the requirement whose 'freedom' is least and to allocate the chosen requirement to the period in which it would cause the least 'interference' to all remaining unallocated requirements. The freedom measure used is the quantity P-N, where P is the number of free periods available for the requirement, and N is the number of assignments still to be made to the timetable for that requirement. The larger the value of P-N, the greater the flexibility in the choice of available periods.

The first program described by Appleby et al chooses the requirement with the lowest value of P-N, and allocates it to the period in which a heuristic 'interference' measure is minimum. Interference is caused to other requirements by the fact that when the chosen requirement is allocated to a period, that period will no longer be free for other clashing requirements. The interference is calculated from the reduction in the 'P' value for every requirement caused by the allocation of the chosen requirement. The contribution of each requirement to the total interference is weighted by the value of P-N of that

requirement in such a way that the greatest contributions are made by requirements whose P-N values are small.

The second program considers P-N values of the following sets of requirements:

a) All requirements involving a particular class
b) All requirements involving a particular teacher
c) All requirements involving a particular set of classes

The definition of P-N for a set of requirements is the same as that for a single requirement, except that a 'free period' is a period in which at least one requirement does not clash and 'N' is the total number of assignments in the group still to be entered. A set for which P-N=0 is a 'tight set' as defined by Gotlieb (section 3.2.2).

The third, untested, program examines available periods for classes whose P-N values are zero. For each such period, it calculates the quantity 'M-C' where M is the number of teachers available and C is the number of classes which do not have as yet any lesson allocated in that period. If there are not enough teachers the timetable cannot be completed.

Tests showed that the methods described were effective, but some problems were experienced in achieving good distribution.

The heuristic method of Berghuis et al (1964) calculates for each teacher, a difficulty or 'resistance' value, which is a complex function of the number of teaching periods required, the number of periods for which the teacher is available, and other similar parameters. Requirements are chosen on the basis of high 'resistance' of the teacher involved. Allocations are made to periods with minimum 'M-C'. The program has a subroutine to interchange timetable entries when necessary, but the authors comment that an experienced human timetabler can make these interchanges more efficiently than this subroutine.

Barraclough's program (1965) starts with a partially complete timetable, and enters blocks and double period requirements before the singles. The first free period found in a scan of periods is chosen for allocation. If a single cannot be fitted by this means, attempts are made to enter it by moving one other clashing assignment already in the timetable to a different period. Barraclough also gives criteria to assess the success of computer constructed timetables, and discusses the problem of allocating subjects to blocks.

Lazak's method (1969) is to assign requirements to periods by a straight-forward 'first free period available' procedure until no more can be entered. Further entries are then made by displacing previous assignments out of the timetable. These displaced entries are reallocated at a later stage of construction Simple heuristic formulae are used to choose requirements to be entered in this manner and to choose the periods into which each requirement is to be assigned. The heuristics favour the displacement of assignments which have been entered early in the construction, which have not been displaced and rescheduled before, and which have not themselves been entered by this displacement method.

The heuristic method of Brittan and Farley (1971) selects each requirement on the basis of minimum P-N and assigns it to the first period found which satisfies the following conditions:

a) if the requirement is for a double or a multiple-period lesson, then the lesson must not cross a break;

b) the period must be allowed and no clashes must be present;

c) the lesson must not be on the same day as any other lesson from the same requirement. If all days have been used, then it must not be on an adjacent period.

The order in which periods are scanned is designed to achieve good distribution. If a lesson cannot be assigned to any period, then an interchange strategy similar to that used by Barraclough (1965) is used. The period chosen for making the displacement is the one with the fewest clashing assignments.

Because of the differing characteristics of the university timetabling problem, relatively straight-forward methods can be used for the basic allocation procedure. Provision must however be made for the more complex conditions that may occur. Almond (1966) describes two algorithms for University timetabling, one for a single department, the other for a faculty of several departments. Both algorithms select lectures and allocate them to the first period found that satisfies the special conditions imposed by the University on the time patterns of lectures. Extensions to Almond's algorithm are given by Yule (1968). These extensions enable further conditions and constraints to be satisfied. Almond (1969) also gives a modified version of the originally proposed algorithm. Some of the constraints that these programs cater for are similar to those that appear in school timetabling. Although the basic algorithm is inadequate for the 'tighter' school problem, the constraint-handling techniques are applicable.

The computer preparation of examination timetables has been investigated by Cole (1964), Broder (1964) and Wood (1968). The basic method is to form a conflict matrix indicating which pairs of examinations may not be scheduled together, and to allocate the examinations using a simple straight-forward procedure, taking into account any special conditions. Cole's (1964) method allocates the examinations period by period. For each period it chooses an examination which satisfies all of the conditions and which does not conflict with any examination already in the period.

Broder's method (1964) allocates examinations to a fixed number of periods, and attempts to minimise the total number of conflicts. Broder's conflict matrix gives the number of students who are taking each possible pair of examinations. Each examination is allocated to the period for which the resulting number of students having to take more than one examination in one period, totalled over all periods, is minimised.

In Wood's method (1968) the examinations are sorted into order of difficulty, and are allocated to periods to maintain acceptable distributions of examinations for candidates and to restrict the periods available for other examinations as little as possible. The size of the room required is also taken into consideration, and a room is allocated to each examination assigned to a period.

The conflict matrix technique is also applicable to school timetabling, but because of the greater number of requirements involved the matrix can become unwieldy to manipulate. Unlike examination timetabling it is more practical in the school problem to determine clashes directly by finding whether there are teachers and classes common to two requirements.

3.2.2 The Theoretical Approach of Gotlieb

A timetabling method that has received much interest is that due to Gotlieb (1963). The method is based on a theorem due to Hall (1935) which gives a necessary and sufficient condition for the existence of a system of distinct representatives for a collection of subsets of a set. Gotlieb (1963) represents the timetable as a 3-dimensional array whose dimensions are teachers, classes, and periods. The initial requirements are given as a 2-dimensional array whose elements are the number of single-period meetings required for each teacher-class combination. Also included in the input data are preassignments. Associated with the timetable at every stage of construction is another 3-dimensional array of 1's and 0's - the availability array - which indicates

what periods are available for the scheduling of each class-teacher combination. By taking 2-dimensional sections of this array and applying the Hall conditions to each section, it is possible to show:

a) that the timetable is either probably feasible or definitely infeasible,
b) that certain elements of the availability array can be 'reduced' from 1 to 0, thus reducing the available choice of periods for some of the teacher-class combinations. This is done by segmenting the elements of each array section into 'tight sets'.

The process of changing availability elements from 1 to 0, called 'availability reduction', is applied iteratively to each of the possible 2-dimensional sections in turn, until no further elements are changed. The new availability matrix can be used for choosing a period for scheduling the next meeting. After the meeting has been scheduled the availability matrix is again reduced. The process continues until a solution is found, or the availability matrix shows infeasibility.

Gotlieb (1963) conjectured that if this procedure is followed, a solution will always be found if one exists. However tests of the method carried out by Csima and Gotlieb (1964) showed that this conjecture was false, and proposed a modified version. The modified conjecture states that if a solution exists, then the result of availability reduction after any available period has been chosen is either infeasibility or an availability matrix which contains a solution. Csima and Gotlieb also briefly discuss the relationship between the Gotlieb method and theory on doubly stochastic matrices and bipartite graphs. The theory associated with the method is investigated in detail in Csima (1965).

Duncan (1965) presents further results and running times for the Gotlieb method. The solution time was found to increase by a factor of about 2 for each teacher and class added.

Lions (1966a) applied the Hungarian algorithm of Kuhn (1955) to the matrix reduction problem in Gotlieb's method, to reduce the amount of computation required. Lion's method is essentially to construct a 'schedule' in each 2-dimensional array section by means of a subroutine 'EXPAND' (section 5.2.5). This subroutine will detect whether the availability array is probably feasible or definitely infeasible, and will indicate the elements that need to be reduced or the elements that must not be reduced.

Gotlieb's modified conjecture was shown to be false by Lions (1966b), who found a counter-example. However Lions states that such counter-examples are rare and that the Gotlieb method is still potentially a practical method. The method, with Lion's modifications, was in fact developed into a complete practical system which was used for timetabling for Ontario schools (Lions 1967).

As the basic Gotlieb method could not handle blocks, double period or distribution, extra refinements were needed. The weekly timetabling problem is divided into five (or six) smaller daily problems, by first allocating the requirements uniformly between the days. Then blocks, double periods and other complex requirements are allocated by a heuristic 'look-ahead' program which attempts to avoid blocking out later assignments. The allocated blocks and double periods are considered as preassignments for the remaining process of assigning the remaining single-teacher-single-class lessons using the Gotlieb-Lions method.

A generalisation of the Gotlieb method is presented by Lions (1968). The availability array is generalised to the 'opportunity list' which indicates what meetings (lessons associated with periods) are available for inclusion in the final timetable.

Meetings are selected for inclusion by means of an 'assignment strategy'. A 'minor infeasible condition' occurs when infeasibility occurs after a meeting is selected, and a 'major infeasible condition' occurs if any meeting chosen results in a minor infeasible condition.

Dempster (1968) considers the problem of availability reduction by examination of the full 3-dimensional availability array rather than of planar sections. Necessary and sufficient conditions for an array element to be part of a solution are given for the three dimensional case. It is shown that either an interchange with an existing solution can be constructed, or a 'redundant chain' of availability elements exists through the element in question.

Two algorithms based on this theory are given in Dempster (1971). The first is a graph-recolouring algorithm which nears some resemblance to the tree search algorithm of Oliver (1968). The second is a 'primal-dual' algorithm for a timetable problem in which the sum of weights on the solution elements must be optimised - effectively a three-dimensional generalisation of the assignment problem.

Lions (1971) presents further theory on availability reduction in the special case where a single element of a reduced and feasible matrix has been changed from one to zero. Two efficient algorithms for reducing such matrices are given, as well as an algorithm for constructing the tight set containing a given element of the matrix.

The basic Gotlieb method is applicable only to single-teacher single-class problems. Nevertheless in Chapter 5 it is shown that the method can be applied to teacher-period and class-period schedules of timetabling problems involving blocks. However there is no direct equivalent to the teacher-class section of the availability array and hence the Gotlieb method cannot be applied fully. Chapter 5 gives a new algorithm for constructing period schedules in the more general problem.

3.2.3 Integer Programming

The school timetabling problem can be converted into an integer programming problem by defining variables to represent allocations or patterns of allocations to periods. The values of these variables are restricted to 1 or 0, where 1 means that the allocation is made and 0 means the allocation is not made. The constraints of the timetable problem can be expressed as inequality relations on these variables and a 'cost' function can be defined to represent distribution quality. Standard integer programming techniques can then be applied to find values for the variables and hence a solution to the problem.

Lawrie (1966, 1968, 1969) uses the concept of 'layouts' which is due to Lewis (1961) and which has been introduced earlier in the discussion of manual timetabling. A layout is essentially a complete set of blocks for a form or year group. An 'arrangement' is a set of blocks, one from each layout, which do not clash and which could potentially form a period in the final timetable. All possible arrangements are found and serially numbered. A timetable consists of a set of arrangements which satisfies the school's lesson-number requirements, which can be expressed as:

$$\sum_{j=1}^{N} a_{ij} x_j = b_i$$

where

a_{ij} = 1 if block i appears in arrangement j
= 0 otherwise

b_i = the number of occurrences specified for block i in the layout,

x_i = the number of occurrences of the jth arrangement in the timetable.

In earlier work an objective function of the form

$$\text{maximise} \quad \sum_{j=1}^{N} c_j x_j$$

was used, but later versions of the program dropped the objective

function in favour of generating a number of feasible solutions by means of an integer programming algorithm in conjunction with an ad hoc procedure. The execution times range from about 1 to 12 minutes with various data sets consisting of 28 to 48 blocks and 210 to 2400 arrangements. However it is difficult to satisfy distribution and double-period constraints with this method. The periods in the timetable produced must be permuted to satisfy these conditions to a reasonable extent. Also, as mentioned before, the layout scheme is not suitable for the timetabling problems of many schools. Nevertheless the method could in principle be used to determine whether the blocks in the school's requirements could be fitted into the timetable without distribution constraints, and hence to indicate the need for changes in the teacher allocations within the blocks.

Akkoyunlu (1973) presents a linear programming algorithm for a university timetabling problem, which handles weighted constraints A variable is defined for each possible allocation of a course to one of a number of alternative time schedules. Incompatible assignment pairs are represented in a set of ordered pairs of the variables. A cost function can be defined to indicate the desirability of any particular assignment, and to indicate the interaction between any pair of assignments.

The constraints on the assignments can be expressed as inequalities and a linear sum can be defined to express the total cost which must be minimised. A modified simplex algorithm is shown to give the desired 0-1 values for the variables. However the number of variables can be quite large even for modest sized problems. If there are 30 courses, each with a choice of about 6 or 7 alternative time schedules, there will be some 200 variables, and almost the same number of inequalities. For a problem of the size of the school timetable problem, the corresponding linear programming problem would be impractical.

3.2.4 Tree Searching

An alternative approach to timetable construction is to backtrack from infeasible situations, rather than to try to avoid infeasibility. A tree structured search can be applied to the timetable problem in such a way that a solution can always be found if one exists. A simple but naive method is defined by the algorithm of figure 3.1. This method allocates requirements to periods on a 'first available' basis. If an infeasible situation is reached, backtracking consists of removing the last assignment entered and allocating it to the next available period.

(Assume $N_r = 1$ for all r, for simplicity. No loss in generality is incurred, since a requirement r with $N_r > 1$ can be replaced by N_r requirements $r_1, r_2, \ldots, r_{N_r}$ each with $X_{r_i k} = X_{rk}$ and $N_{r_i} = 1$).

Step 1. $r \leftarrow 1$

2. $p \leftarrow 1$.

3. Test r in p.
   If p is not free for r, go to 4.
   If p is free for r, enter r into p and go to 6.

4. $p \leftarrow p + 1$, if $p \leq p_{max}$ go to 3.

5. This step is reached if there was no period available for r. If $r = 1$ then there was no solution.
   If $r \neq 1$ then
   $r \leftarrow r - 1$
   p ← period which currently contains r
   Remove r from p
   go to 4.

6. $r \leftarrow r + 1$. If $r > r_{max}$, a solution has been found.
   Otherwise go to step 2.

Figure 3.1 Naive Tree Search Algorithm

Deutsch (1966) describes two short cuts to this simple method. The first short cut recognises that requirements which clash with less than p_{max} other requirements can always be entered into the timetable. The second short cut recognises during construction the presence of requirements that clash in n periods ($n < p_{max}$) and also clash with m other requirements not yet entered, where $m + n < p_{max}$. Such requirements can always be entered and hence assignment of these can be left to the end.

In school timetabling these situations in fact rarely arise. Every assignment clashes with the $p_{max}-1$ other assignments with the same class (classes have lessons in all p_{max} periods of the week) together with several other assignments with the same teacher. Thus the first short cut can never be used, and the second can be used only infrequently since usually $m + n > p_{max}$ and only sometimes does $m + n = p_{max}$.

Johnston and Wolfenden (1968) describe a heuristic method which incorporates the tree search algorithm presented above. In order to reduce the amount of backtracking required, the 'tightest' requirement is chosen at each stage for next assignment, as in most other heuristic algorithms.

A different type of tree searching algorithm, and one that is extended and modified in this thesis, is presented by Oliver (1968). Here assignments are made to the timetable until no further can be entered, at which stage a tree search of interchanges is carried out. Cycling is prevented by recording the assignments on a stack. Like the 'naive' algorithm of figure 3.1, this technique is also exhaustive and will find a solution if one exists.

3.2.5 Other Methods

A network flow model of the simple single-teacher single-class problem has been proposed by de Werra (1971). The problem is transformed into a bipartite graph (a graph in which all vertices can be partitioned into two sets, and no edge connects two vertices in the same set). A source and a sink are added.

The source is connected to all vertices of one set and the sink to all vertices of the other. Capacities are defined for each edge in the graph and the resulting flow problem is solved to give a minimal cost integral valued flow. (The cost function gives priority to meetings with the lowest degree of freedom). The solution is a schedule for a period of the timetable. The edges corresponding to this solution are removed from the network and the method repeated to find schedules for the remaining periods.

Like the basic Gotlieb method this method will not handle blocks, double periods, or distribution. Blocks and double periods have to be preassigned before the method can be used. Also the method does not guarantee a solution. For a 34-class 64-teacher 35-period problem with 1200 meetings (approximating Riccarton High School) the computation time was 30 minutes but 3% of the meetings were not assigned.

The timetabling problem (with blocks) is equivalent to the problem of colouring the vertices of the clash graph, in such a way that no pair of vertices joined by an edge have the same colour. The chromatic number of the graph, defined as the minimum number of colours required, is the minimum number of periods into which a set of requirements whose clashes are represented by the graph, can be timetabled.

Welsh and Powell (1967) use a simple graph colouring algorithm to show that:

$$\max_i \min_i \left\lceil d_i + 1 \right\rceil$$

is an upper bound on the chromatic number of a graph, where d_i is the degree of the ith vertex (the number of edges having vertex i as their endpoint). However results given by Wood (1969) indicate that this bound is usually more than twice the true value. Wood's algorithm for graph colouring utilises a 'similarity matrix' for determining which pairs of vertices should be

the same colour. The similarity matrix is defined as follows:

S_{ij} = 0 if vertices i and j are linked

S_{ij} = number of other vertices to which i and j are both connected, if i and j are not linked.

The algorithm is essentially heuristic, in that it colours pairs of vertices with high similarity first. Results given show that when the proportion of edges to vertices in the graph is high, this method is superior to the standard method in which vertices are arranged in decreasing order of their degree.

3.3 Solving the Practical Problem

A variety of methods for school timetabling and related problems have now been introduced. Many of these methods have shortcomings when applied to the practical problem. Theoretical methods break down when the complexities are introduced, or are inapplicable to problems as 'tight' as the practical problem. Integer programming problems generated from real school data are too large and cumbersome or require alternative requirement formulations which can be as difficult to construct as the timetable itself. Exhaustive tree searching algorithms would take impractically long to search all possible arrangements of the lessons in a realistic problem. The most practical timetabling method which can handle all the complexities and which can construct a timetable within a reasonable time is the heuristic method.

To design an effective timetabling system, some further characteristics of the practical problem must be taken into consideration. To produce a partially complete timetable in a single computer run is not sufficient. The timetable must be 100% complete before it can be used. The problem is what to do with the lessons that cannot be fitted in. When he finds that a lesson will not go in, the manual timetabler makes compromises or changes to the teacher allocations to allow the lesson to be fitted, perhaps after some interchanging. Changing a teacher

allocation is a decision which involves many human factors, such as the suitability of the new teacher for the job, and the continuity of teaching for the class involved. It is a decision that cannot be made by the computer. Yet the best time for making the decision is as soon as it is found that the lesson will not fit. If it is left till later the timetable will fill with other lessons and will restrict the options available for compromises. A less satisfactory timetable will be the result.

It follows that a computer system that does not allow human intervention during construction will give a less satisfactory result than one that does. A system that attempts to construct the timetable in one run is likely to pose some very difficult problems for the manual timetabler who has to complete the resulting partial timetable. If a block is omitted, the timetabler has the option of either repeating the run after modifying the block or associated clashing lessons, or entering the block by hand. The first alternative can lead to other blocks being omitted on the second run, while the second alternative necessitat the removal of a large number of singles to allow the block to be entered. All of these singles would then have to be replaced. Obviously the task of entering the block would be much easier if it were done at the time the program failed to get it in.

Even if a 100% fit could be achieved, the computer-constructed timetable may still not be satisfactory for use by the school, even though it satisfies all of the constraints in the input data. This is because some constraints are not easy to specify precisely for the computer. An example of such a constraint is the need to group the teaching periods of part-time teachers. It is easier for the timetabler to examine the timetable during construction to check such constraints than to pre-specify a precise and general rule to cover all possibilities that may occur. Checking and correction of violations must take place at a suitable stage of construction. If checking takes place too early, then later

steps of the construction may disrupt the pattern and cause the constraint to be violated again. If checking takes place too late, it may be too difficult to correct any violation because of the number of other lessons in the timetable. A computer timetabling system which does not allow human intervention is liable to produce timetables with many violations of ill-defined constraints which are difficult for the timetabler to correct. Although the timetable could still be used, such violations only degrade the quality of the timetable.

These characteristics suggest that a good timetable construction system must allow many short computer steps to be interspersed with human activity. Ideally the timetabler should be interacting on-line with the machine, using a keyboard and a CRT or similarly fast display device for efficient communication.

The idea of interactive timetabling is not new. Lions (1968) suggests that timetabling is an interactive computing problem, and Ryan (1969), after surveying existing timetabling methods, concludes that the problem should be shared between timetabler and computer. The interactive method described in this thesis was developed from a system proposed by Higgens and Andreae (1969). An early stage in the development of this method is described in Andreae et al (1972).

CHAPTER 4

THE INTERACTIVE APPROACH

4.1 Introduction

This chapter traces the development of the new interactive approach and presents the display formats and interaction techniques that were finally successful. After a brief overview of the basic structure of the interactive system and the hardware environment in which it was developed, attention is turned towards the design of display formats. The shortcomings of earlier formats are discussed from the point of view of interaction efficacy. The advantages of the final designs are given. Although the display formats presented appear highly complicated, the final designs have been tested in practice and have been proved to be effective.

The man-to-machine side of the interaction is then considered. The timetabler can carry out either basic operations or a composite set of operations on the timetable. It is necessary to allow wide flexibility in the choice of composite operations and it is shown how this can be done in a simple program.

The problems of interaction design are discussed and general conclusions are drawn from the experience gained during the development of the interactive approach. To support these conclusions, the general theory of problem-solving proposed by Newell and Simon (1972) is applied to timetabling.

4.2 An Overview of the System

The hardware used during the development of the system consisted of:

a) an EAI 640 digital computer with a 16K core memory of 16 bit words,

b) a Tektronix 611 storage tube display 'scope,

c) a Tektronix hard copy unit,

d) a high-speed paper-tape reader/punch unit,

e) a KSR 35 teletype,

f) a 300K word disc, for storage of program overlays and timetable data files,

g) a DECTAPE magnetic tape unit, used as a disc dump.

A block diagram showing the structure of the interactive system is given in figure 4.1. Hardware not essential for interaction has been excluded from this diagram.

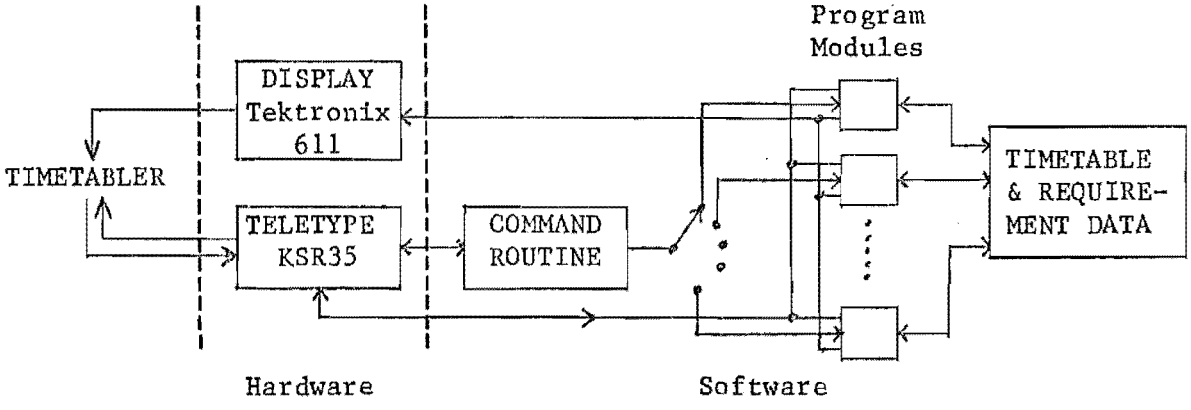


Figure 4.1 Structure of the Interactive Aid

The program is structured as a set of modules. Each module contains code for generating a display on the scope or for performing a simple or a complex operation on the timetable (via subroutines which are not shown). A central command routine selects one of these modules for execution in response to a two-character command typed on the teletype keyboard by the timetabler. After completing its function, the module returns control to the command routine, ready for the next command. This structure permits the timetabler to carry out operations or generate displays in any sequence he wishes.

The initial data detailed in the block timetable (figure 2.11) is stored in the form of a list of requirements (figure 4.2).

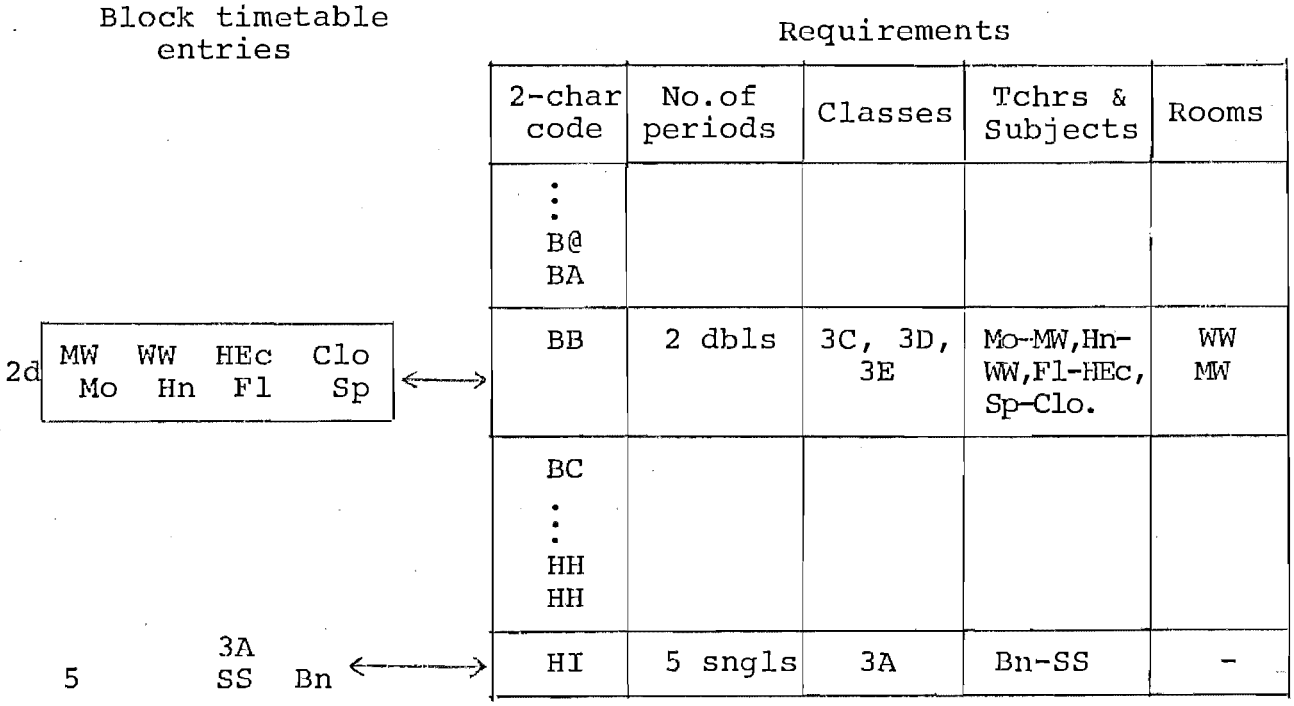


Figure 4.2 Requirements

Each requirement contains teacher, class, room, and period-number information as was described in section 2.4. Additional information is included for each requirement to indicate whether single- or double-periods are needed (both types may be included in one requirement), the subjects taken by the teachers, and restrictions on period availabilities and distribution patterns. Distribution constraints will be described in more detail in Chapter 6.

For identification purposes each requirement is labelled with a two-character code. In earlier versions of the program, the code was a hexadecimal numbering system which used the characters @,A,B, C,..O to represent 0,1,2,...,15 respectively. A code referred directly to the position of the requirement on the requirements list. In later versions of the program, the coding system was changed for reasons to be given later in this chapter.

This interactive system was developed from the manual method, rather than from a non-interactive timetabling program. At first, timetables were constructed manually with the computer providing only the very basic operations of clash-checking and printout. Then sections of the manual process were automated to reduce tedium and effort. The benefits which resulted from taking this approach are as follows:

a) A good 'feel' for the timetabling problem was obtained by constructing the timetable manually in the early stages of the project.

b) Sections of the manual method were automated only where it was found advantageous to automate them.

c) All aspects of the manual process were catered for in the system that evolved.

d) With manual backup constantly available, it was possible for Riccarton High School to entrust the construction of their timetables to the computer aid. This enabled valuable experience of interactive timetabling to be gained without the danger of failure to produce a timetable for the school.

4.3 Design of Displays

The function of the display modules in the system is to generate displays on the screen in response to appropriate commands from the timetabler. The display modules do not alter the timetable or requirement data in any way. Their sole purpose is to provide information to the timetabler about the state of the timetable.

The design of the display formats must be such that the timetabler is able to obtain the information that he wants within a reasonable time. To make progress in construction, he must be able to obtain the right information quickly.

Lack of understanding of human behaviour in timetabling has made the questions of what is the 'right' information and how it can be obtained quickly difficulty to answer. A considerable amount of experimentation was therefore necessary in display designing.

A practical constraint that had to be contended with was the size of the display screen. A screen of unlimited size would obviously greatly simplify the problems of display design. However in most of today's CRT display devices there is a severe restriction on the amount of information that can be presented on the screen. This acted as a further influence in the choice of display format.

4.3.1 Displaying The Whole Timetable

Initially, attempts were made to display the whole timetable on the screen in a format similar to that of the manual board (figure 4.3). In this format, classes are listed down the left-hand side and each period is a column 2 characters wide. Each assignment in the timetable is represented by a block of characters whose width is one period (2 characters) and whose height and position are arranged to cover the classes involved in the assignment. If the lesson is a double-period, then the block of characters is repeated in the next adjacent period. At the top of the character block is the two character numbering code of the associated requirement. Below the requirement code are listed the teachers, represented here by letter-number codes to avoid confusion with the requirement codes. Arrows are used to fill the remaining space to ensure that the classes involved are covered.

This display as designed for the 611 screen is virtually unreadable because of the extreme compaction necessary. It is surprisingly difficult to identify the individual assignments. The periods need to be spaced apart and clearly visible divisions should be present between the assignments in a period. Ideally, each assignment should appear as a unit clearly separated and distinct from its neighbours. This however would require a larger and much more expensive display device.

Figure 4.3 Whole Timetable Display

The timetable display shown in Figure 4.4 is an improvement as far as identifying individual assignments is concerned. The large blocks, such as @I, stand out relatively clearly. For each class the display gives the code of the assignment for that class in each period. Repetition of the code vertically indicates the extent of the block. This display has been found invaluable in giving immediate information on sizes and positions of blocks, their distribution patterns, and the general 'fullness' of the timetable.

A corresponding teacher display (figure 4.5) gives assignment codes for each teacher. Unlike the class display of Figure 4.4, this one does not present the blocks as spatially compact units since any set of teachers can be involved in a block. However, the display does give information on teacher free-period distributions and on the fullness of the periods for individual teachers.

Neither of these two displays gives full information on clashes. One of the aims in developing the computer aid is to eliminate the visual scanning associated with finding clashes on the manual board. However, no format for the whole timetable can give clash information for every assignment in every period such that no visual scanning is required.

4.3.2 Displaying Portions of the Timetable

The limit on human short term memory (e.g. Miller, 1967) together with the limited extent of the human visual field suggest that the timetabler can only examine a part of the timetable at a time. The visual field limit in particular suggests that one could avoid the small screen problem by using the screen as a 'window' on a larger display of the complete timetable. This necessitates the provision of mechanisms for moving the window to different parts of the display under the direction of the timetabler. The operation of such mechanisms is likely to be considerably slower than the visual scanning processes used in examining the complete display on a manual board. Also the window excludes peripheral

Figure 4.4 Class-Period Display

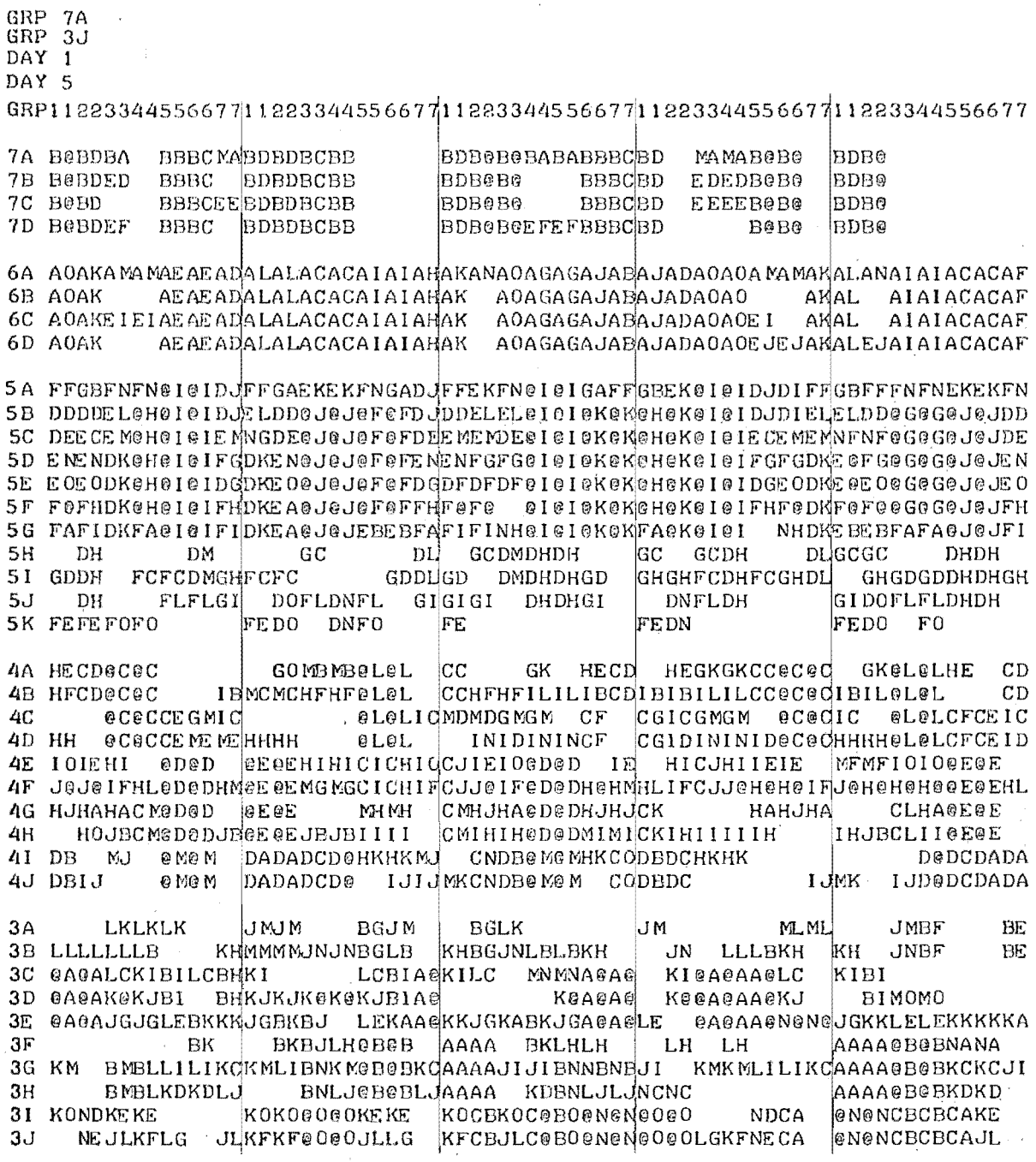


Figure 4.5 Teacher-Period Display

information which can be of importance in locating position. Thus the 'window' technique was considered unsuitable for timetabling, and no development was carried out in this direction.

4.3.3 Requirement Display

A better approach to the problem of determining what portion of the timetable to display is to consider the nature of the task. Most of the time the timetabler is concerned with entering assignments into a partially filled timetable. He must first look for a period for the assignment in which there are no clashes and which is suitable as far as distribution is concerned. This period must also be one permitted for the assignment. If no such period is found, he must choose a period in which the clashes appear reasonably easy to shift elsewhere. Having chosen such a period, he must turn his attention to the displaced clashes.

The above observation suggests that the portion of the timetable displayed should be 'all information for one requirement only'. A display for one requirement only allows a clearer representation of information about the requirement than does a display of the whole timetable. The information that we wish to display consists of the following:

a) Distribution pattern of assignments associated with the requirement.
b) Positions of free periods.
c) Disallowed periods.
d) Clashing assignments in each period.

(a), (b) and (c) can be represented by the use of easily recognizable symbols in the appropriate period positions.

The representation of the clashes is less easy, even though a single-requirement display permits clashes to be identified in

a period without extensive visual scanning. In an early version of the display (figure 4.6) the clashes were represented by teacher lists in a format similar to that of Figure 4.3, classes being included for the teacher clashes. The intention was to give information about the clashes similar to that which the manual timetabler obtains from his board. Since the manual timetabler coped satisfactorily without two-character codes, it was felt at that stage that these codes could also be eliminated from the computer displays as an unnecessary complication.

The reverse was found to be true. The display became usable only when the lists of teachers were replaced by the two-character codes (figure 4.7). This experience illuminated the importance of the two-character codes for effective interaction.

The reasons for the importance of two-character codes appear to be:

a) A code with a minimum number of characters is needed to specify requirements to the computer quickly and unambiguously via the teletype keyboard.
b) Codes allow more compact displays than do lists of teachers' names.
c) Consistency. If codes are employed to specify requirements to the computer, then they should also appear on the displays.
d) Two-character codes are easier to remember than codes with more characters.

The 2-character code based on the hexadecimal numbering system tended to be difficult to 'learn'. Association of a code with the requirement it represented was not easy. Consequently, the code was changed to one in which any two characters could be used, apart from certain characters which had special significance in the commands. The only restriction was that the pair of characters be unique (different from every other pair used for requirements, teachers or classes). Letters, numbers and other

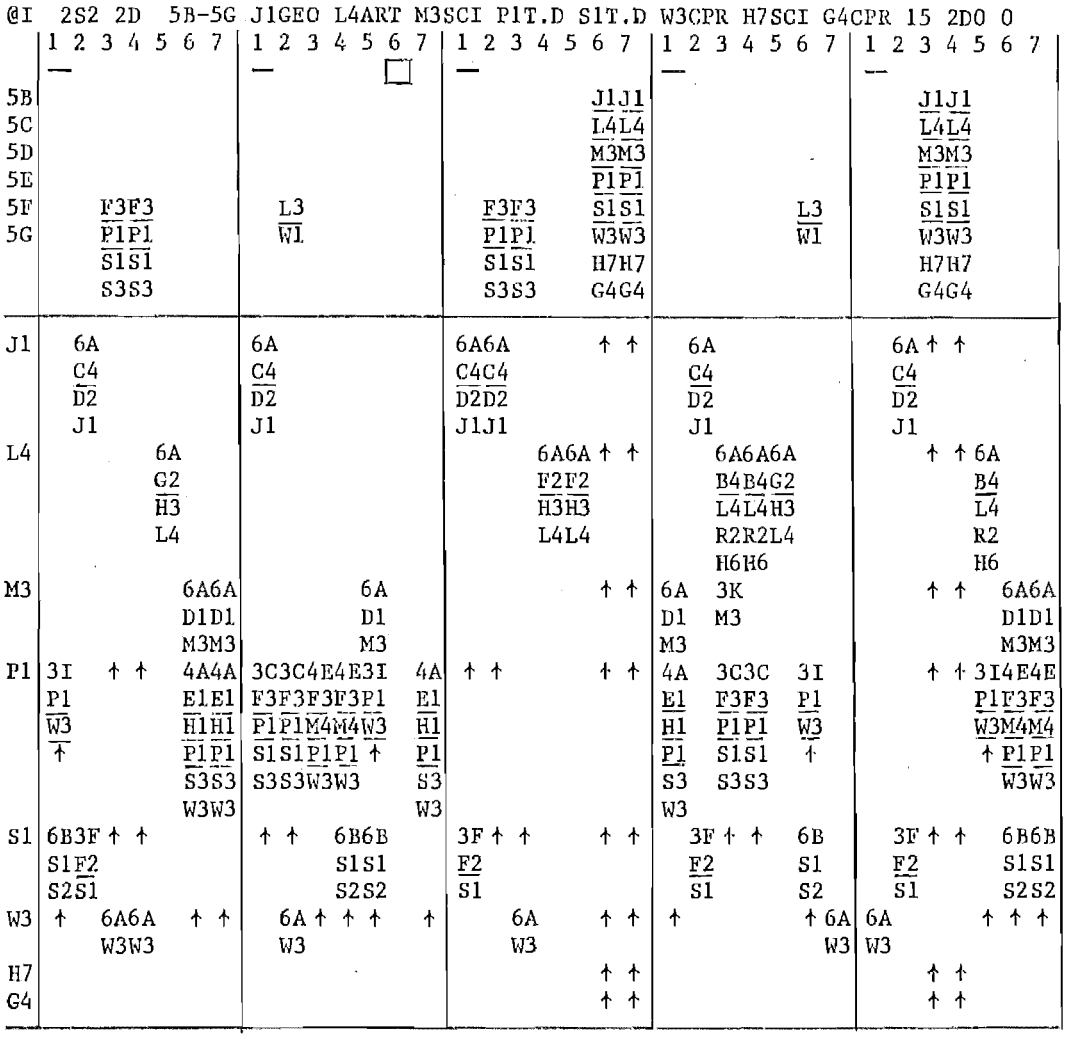
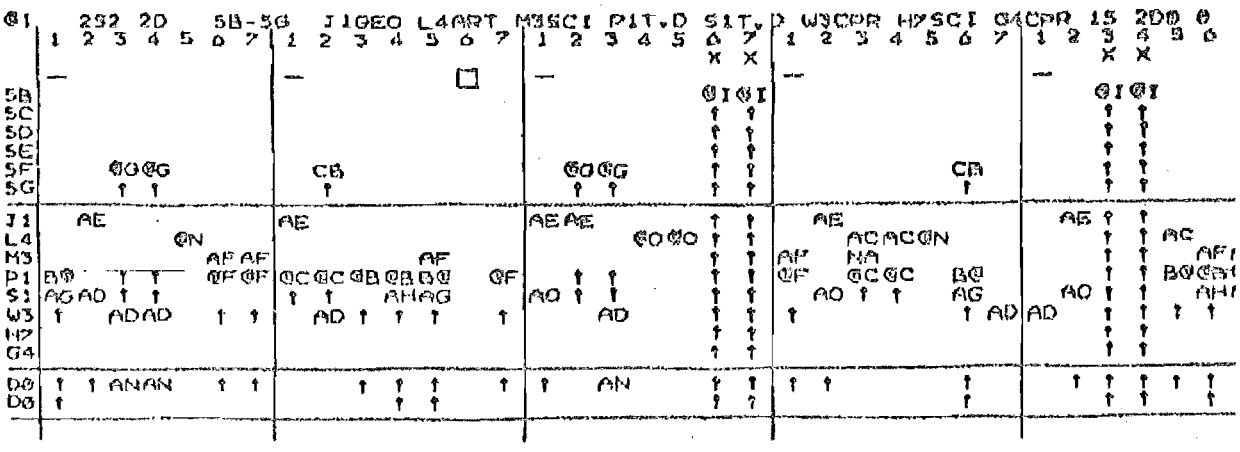


Figure 4.6 Early Version of Requirement Display

Figure 4.7 Requirement Display with Two-Character Codes



teletype characters were allowed. This new coding system facilitated the use of conventions for naming requirements, such as:

Letter-number pair	- single-teacher-single-class lesson (lette corresponds to class, number to subject).
First character = "+"	- 6th form block
First character = "&"	- 5th form block
First character = "@"	- 4th, 3rd Woodwork and Metalwork blocks
First character = "#"	- 4th, 3rd Typing blocks

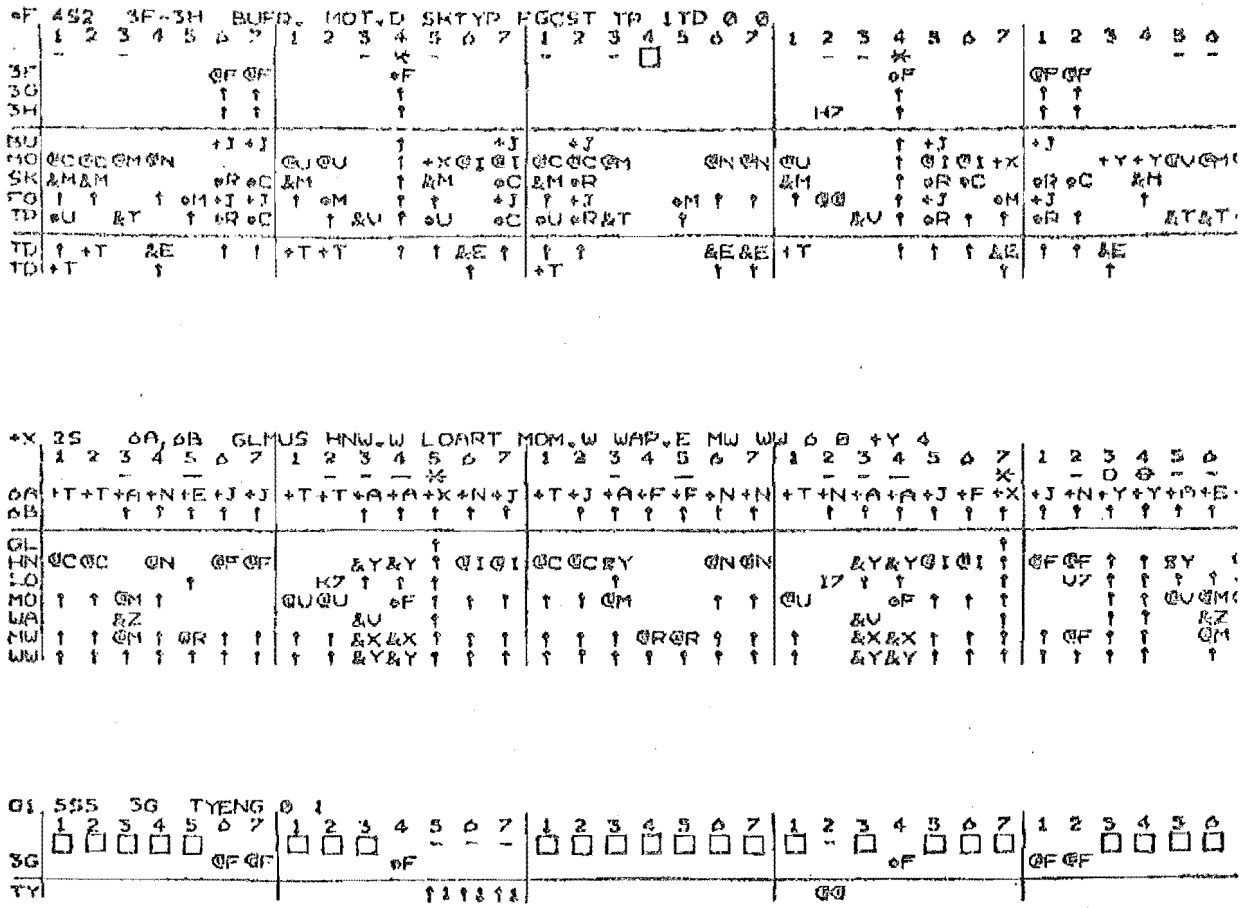
Such conventions greatly simplified the learning task. On the individual requirement displays it became easier to recognise whether a clash was a block or a single and what kind of block it was. The new code also allowed two-letter teacher abbreviations to be employed instead of the original letter-number combinations which were necessary to avoid confusion with the old two-letter requirement codes.

Examples of the final version of the display for a single requirement are shown in Figure 4.8. In detail, the information provided comprises:

1) The top line gives details of the requirement, viz the code name, the numbers of singles and doubles, the classes, the teachers with their subjects, the classrooms required, reference to forbidden periods and distribution information. This is the system's standard format for representing a requirement. (See Appendix C.)

2) Below the period numbers, the distribution is shown first of the requirement itself and then for each 'distribution group' in which the requirement is involved. The need for distribution groups arises because the periods of a particular subject can originate from more than one requirement. All such requirements must be considered when examining the

Figure 4.8 Examples of Final Form of Requirement Display



distribution of the subject. The positions of assignments associated with the requirement displayed are shown by X's while the position of other assignments involved in distribution groups are shown by O's.

3) The positions of non-clashing periods are shown by □'s.

4) Periods forbidden for the displayed requirement are shown by long dashes - "—".

5) The classes, teachers and classrooms involved in the requirement are listed in the left-hand column; alongside each is given the code of the assignment involving that item in each period of the timetable. Arrows indicate that the code is the same as the one immediately above. If a clashing assignment has been preassigned and must not be moved, a short dash "-" appears in the row immediately above the clashes.

The days are separated by vertical lines. The space between characters in a pair within a period is smaller than the space between characters in adjacent periods.

With this display the operation of finding a free period becomes one of typing the command to generate the display and visually locating the symbol □ . The time taken for an experienced person to do this on average is about 6 seconds composed as follows:

Time to type command	2.5 seconds
Time for computation required for display generation	1.5 seconds
Time to locate position of period visually	2.0 seconds
	6.0 seconds

This is only a slight reduction on the time of 6-15 seconds required for the equivalent all-manual process of scanning a

period to locate a clashing teacher. However, whereas the time for all-manual clash-checking increases as the number of teachers in the lesson to be entered is increased, the computer display, once generated, gives clash details for ALL periods of the week. Thus, if the clash-checking is part of a search for a free period, the computer-assisted method is considerably faster than the unassisted manual method.

Further, if there are no free periods and a clashing assignment has to be moved in order to allow the new assignment in, then by typing the display command together with a 2-character code of the clash the timetabler obtains the new display immediately below the original one, with period columns aligned. He can then examine the new one and refer back to the original display at any time.

The facility of allowing several requirement displays to be present simultaneously on the screen is particularly important since it gives back to the timetabler some of the ease of switching from one requirement to another, a characteristic of the manual board. In particular, it allows him to retrace his steps after an unsuccessful search and it facilitates searching for 'swaps' (e.g. assignment A from period p1 to p2, assignment B from p2 to p1), which appear frequently in a successful series of moves. The simultaneous presentation of related displays can contribute to effective and productive interaction. Another advantage of the use of two-character codes is that the resulting smaller size of the requirement display facilitates simultaneous presentation.

Finding periods in which to enter assignments is not the only use of the requirement display. If teacher allocations in a requirement have to be changed, the formatting of the clashes alongside teachers and classes allows the timetabler to assess the effect on the timetable of changing a teacher allocation. (This will require a simultaneous display for the new teacher).

Frequently, the display is used for relatively simple tasks, such as checking the presence or absence of an assignment in a period, or observing which periods are forbidden. Another important application of the display is in checking that an operation on the timetable has been carried out correctly.

4.3.4 Teacher/Class Display

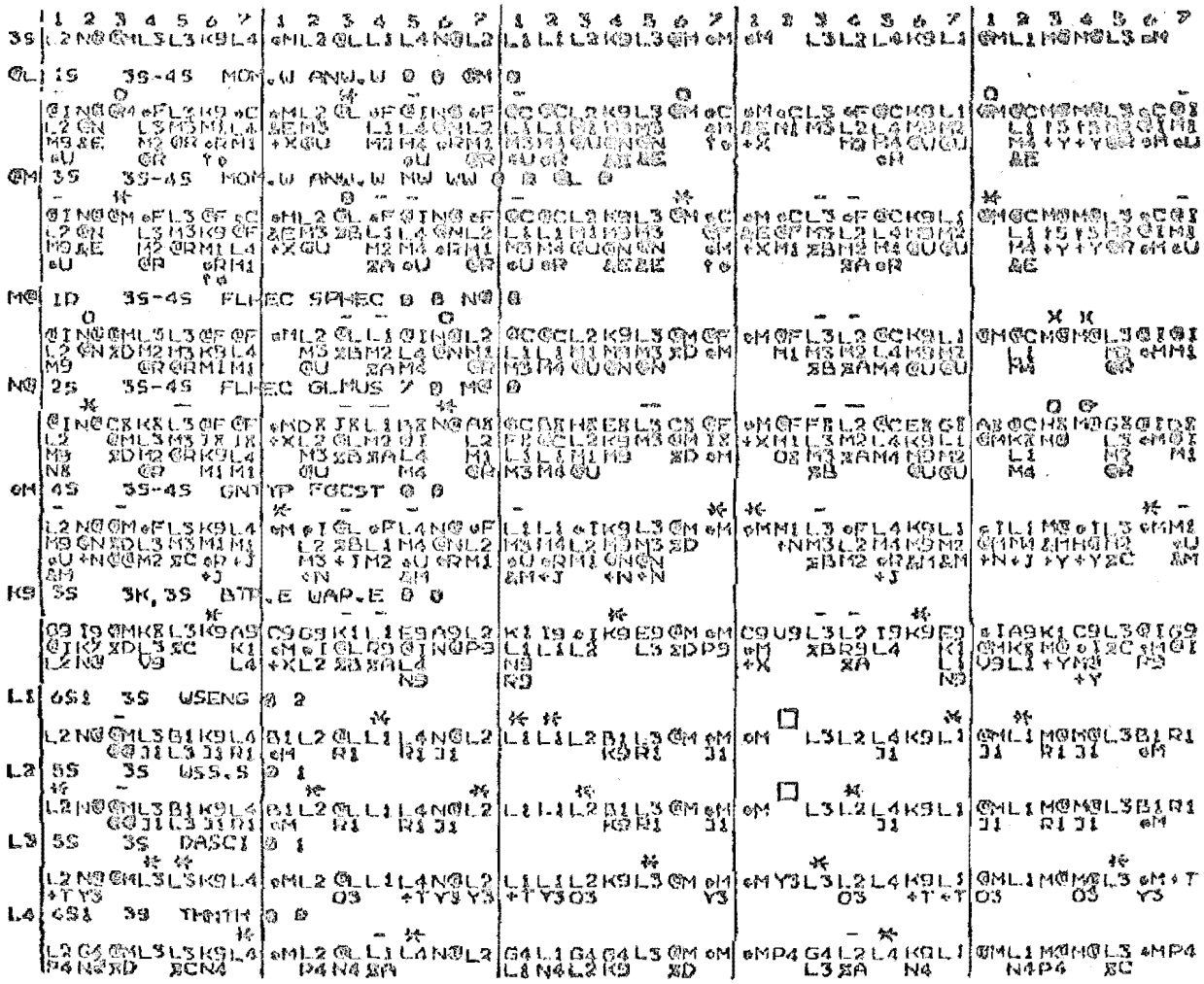
As well as a requirement display, the final system includes a display giving all information for a teacher or a class.

The need for a class display arose from the observation that a special strategy is employed when the last assignment for a class is to be put in. When the class has 34 of its 35 periods filled, attention must be paid to the last unfilled period. Any series of moves that will allow the last assignment in will also fill the empty period with one of the class's assignments. Therefore, the timetabler must be able to examine the clashes in this period for all requirements that involve the class. He will also want to be able to see how to move these assignments around so that the last one can be entered and so that a suitable assignment is put into the empty period.

An example of a class display is shown in Figure 4.9. The format of a teacher display is similar. The display gives the following information:

a) At the top the codes of the assignments involving the class in each period of the timetable.
b) For each requirement involving the class:
   i) The details of the requirement in standard form.
   ii) The distribution, free periods and forbidden periods as described in (b), (c) and (d) for the requirement display.
   iii) The codes of the clashes listed in a column for each period. To minimise space requirements, no attempt is made to format the clashes alongside teachers and classes.

Figure 4.9 Display for Class 3S



A teacher/class display identical to Figure 4.9, but excluding the clashes, has also been included in the final system. This display is smaller in physical size and requires less computation for its generation. Its main functions are to enable the timetabler to check distributions of the requirements of a class, and to observe the positions of free periods or disallowed periods for the class.

4.3.5 Other Displays for Showing Clashes

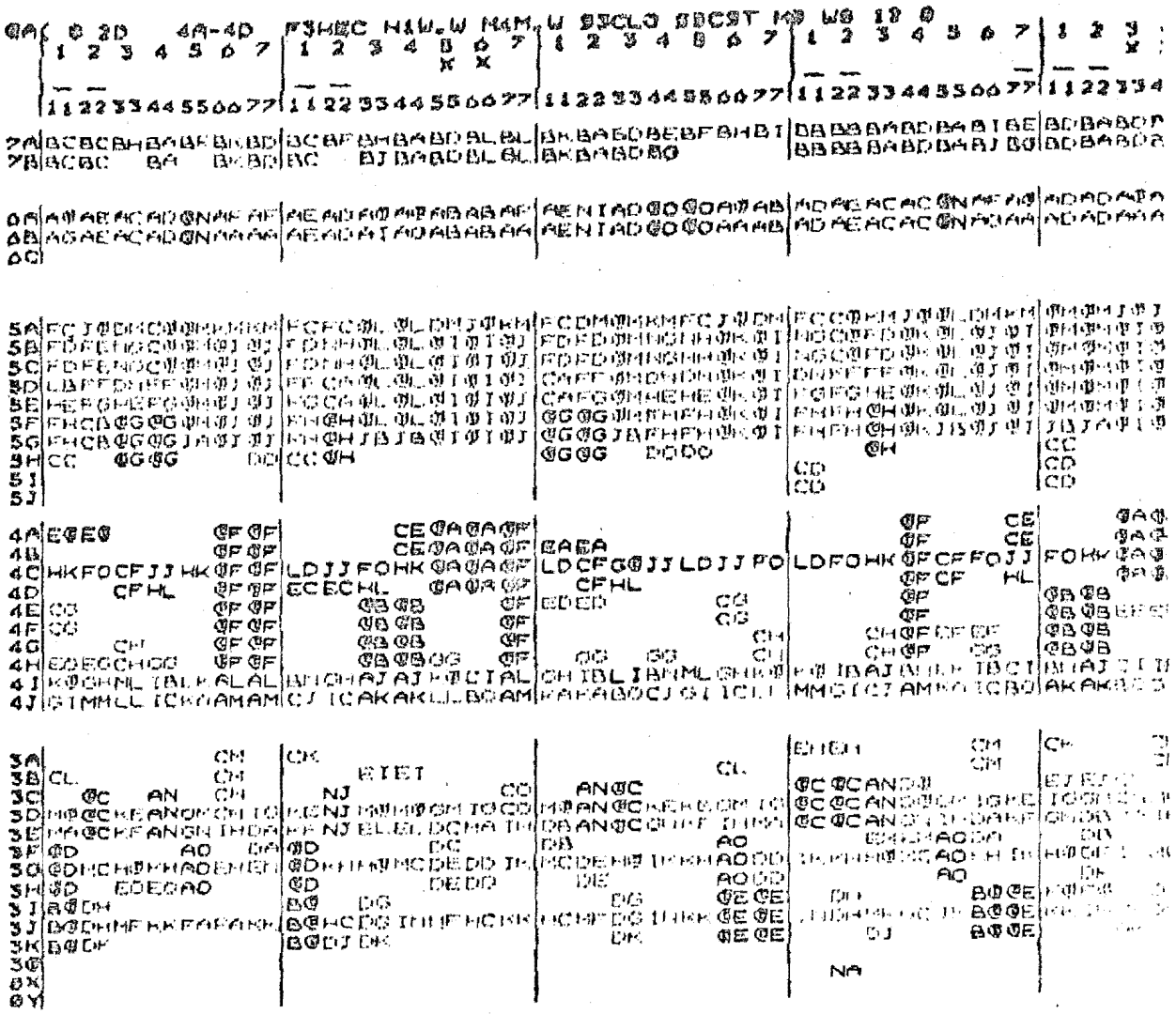
Although the requirement display (figure 4.8) is compact and gives the identities of the clashes, it was felt that more information could be given about the clashes to reduce the need for the timetabler to generate new displays. In an attempt to give extra information two further display formats were tested.

The first display format tested, figure 4.10, attempted to combine the advantages of the single-requirement display (figure 4.8) with the class timetable display (figure 4.4). The intention was to give information on size and positions of the clashes, similar to that which could be obtained directly from the manual board.

The display format was identical to that of the class timetable (figure 4.4) except that the clashes of the requirement specified by the timetabler were 'emphasised'. This was done by 'dotting' the characters of the non-emphasised clashes, while writing the characters of the emphasised characters normally. As a result the clashes 'stood out' from the background of dotted characters.

It was found that this display offered no real advantage over the requirement display (figure 4.8). It reintroduced the need to carry out visual scanning to find the clashes, and it did not provide sufficient information about 'clashes of the clashes' to enable the timetabler to carry out a deeper search for interchanges. The physical size of this display prevented several such displays from being present simultaneously.

Figure 4.10 Class-timetable Display with 'Enhanced' Clashes



In the second display format tested, an alternative approach was taken to the problem of displaying information about clashes. This was to show whether or not the clashing assignments themselves clashed with one another. Suppose assignment A is in period p_1 and assignment B is in p_2 (figure 4.11(a)). If A and B both clash with the specified requirement R, but they do not clash with each other, then B can be moved to p_1 and R inserted in p_2. However if A and B clash with each other, this operation cannot be performed. Thus it appears that information on clashing between clashes could be useful in determining whether the specified requirement could be entered.

A and B clash only if they involve at least one common item. If we display in one row the assignments in the timetable involving one of these common items, the clash between A and B becomes apparent as the presence of the symbols A and B on the same line (figure 4.11(b)). By suitably choosing items, we can display clash relations amongst a number of assignments. Figure 4.11(c) shows immediately that A clashes with B, B clashes with C, but A does not clash with C.

The 'clash' display format is shown in figure 4.12. It is similar to the requirement display format of figure 4.8, but the items have been selected to show whether or not the assignments clashing with #F themselves clash with one another. Assignments in the display which do not clash with R are shown as dashes (-).

This display became too complex, however, when the number of assignments in the timetable was sufficient to introduce difficulty. The time taken for the timetabler to understand the clash relations presented in this display was considered to be too great for the benefits achieved. It was therefore concluded that the clash display, like the 'enhanced' class-period display of figure 4.10, offered little advantage over the requirement display of figure 4.8.

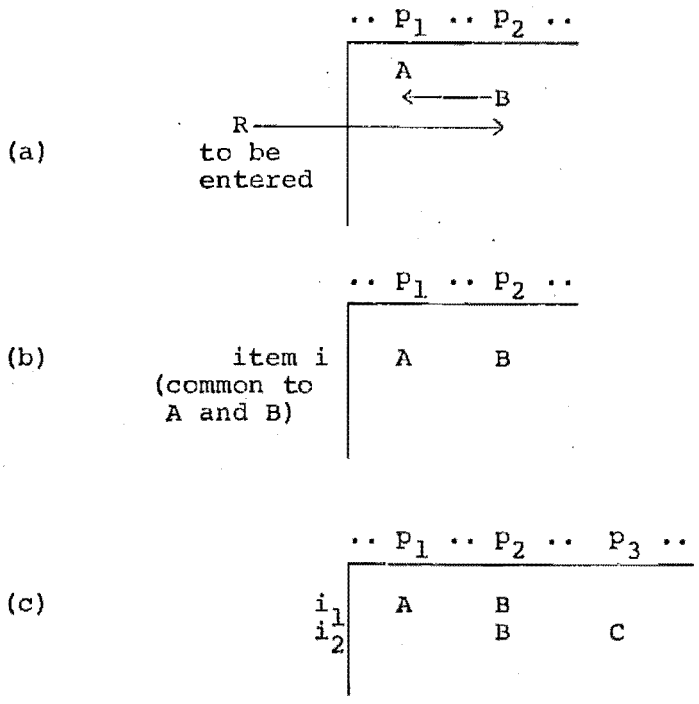
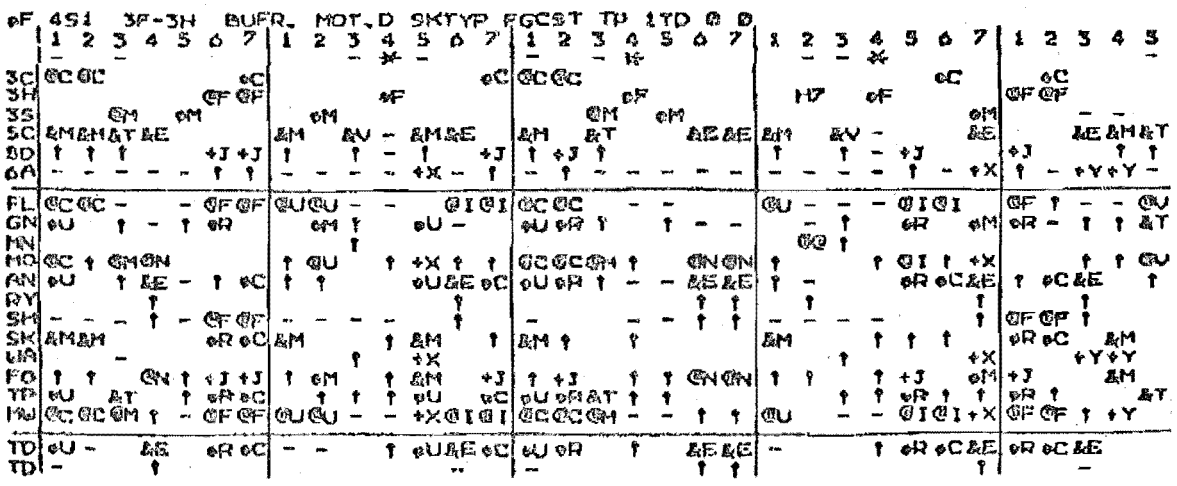


Figure 4.11 Displaying Clash Relations

Figure 4.12 Clash Display for #F



4.3.6 Other Displays for Timetabling

For some purposes, a limited amount of timetable information for a number of requirements or items is required to be displayed simultaneously. As it is not usually possible to fit more than one item display of the format shown in figure 4.9 on the screen, the information given for each item of a set of items must be restricted to the timetable assignments only, that is, the line immediately below the period numbers in figure 4.9. The display for a set of items therefore has the format of the 'clash' section of the requirement display of figure 4.8.

Similarly it is not possible to fit more than a few full requirement displays on the screen. The size of the requirement display can be reduced by adopting the 'compact' format for clashes as shown for each requirement in Figure 4.9. The display for a set of requirements has the appearance of the class display in Figure 4.9, excluding the two top lines.

Both of these displays require the timetabler to specify sets of requirements or items via the keyboard. To use such displays effectively, he must be able to specify the sets as quickly and simply as possible. The need for easy specification of sets occurs in other areas of interactive timetabling and hence set defining is handled by a distinct subsystem.

4.3.7 The Set Defining System

The basic method of specifying a set is to name each member individually. However, frequently the timetabler wishes to specify a set of requirements or items having a common attribute. For example, he may wish to display all requirements involving teacher x, or all items available in Monday period 2. He therefore should be able to define the set simply by stating the attribute, rather than by naming every element.

The set defining system allows the timetabler to specify sets either by naming individual elements or by giving attributes such as items involved or subjects taught, or by giving numeric relation conditions, such as 'all requirements with at least one double-period lesson remaining'.

The system also facilitates the definition of a set which is the union, intersection, or complement of sets specified by the above means. This permits sets to be defined such as 'all requirements in which Art is taught and which have at least one double-period lesson remaining' if Art doubles are to be selected.

The system comprises a subroutine and a special 'language' for defining sets. A program module which requires a set to be defined accepts a string of characters from the keyboard and passes the string to the set-defining subroutine. The subroutine then interprets the string and indicates the defined set by flagging entries in a list of all elements of the appropriate type. The special 'language' used in the string from the keyboard is defined in Appendix A.

A simple extension to the subroutine enables it to define sets of timetable periods. This is necessary for composite load/unload operations (section 4.4.1).

4.4 Timetable Operations

Timetable operation modules modify the timetable or requirement data according to commands entered by the timetabler. They constitute the 'man-to-machine' side of the interactive communication link and as such must allow the timetabler to do what he wants to do quickly and with a minimum of effort.

4.4.1 Load/Unload

The system includes commands which enable manipulation of individual timetable entries. The functions of these basic commands are:-

a) Load an assignment from a given requirement into a given period of the timetable. Before carrying out this operation, the program checks for and lists any clashes that may be present. On a 'go-ahead' signal from the timetabler, the program removes the clashes and enters the new assignment.

If the timetabler indicates 'no-go', no action is taken. This facility eliminates tedious replacement of clashes if an error is made in typing the command and a large number of clashes would be displaced by immediate loading.

b) Unload a specified assignment from a given period.

c) 'Fix' a specified assignment in a specified period so that it cannot be unloaded or moved before being unfixed.

d) Unfix a specified assignment in a specified period.

Frequently the need arises in interactive timetabling to carry out a multiple load/unload operation, such as 'attempt to load all assignments of a given class into available periods' or 'unload the whole timetable'. The detailed specification of such an operation can vary but execution of any multiple-step operation with a single objective is long and tedious if only basic commands are available. It is important, therefore, to provide commands for composite operations.

Composite load and unload commands are provided by using the set-defining routine of section 4.3.7 to specify sets of requirements and sets of periods. The operations are carried out on all periods of the period set for each requirement of the requirement set. In this way, the only details that need to be specified as 'arguments' for these operations are those related to the objective of the operation. For example, if one wishes to load all assignments of a given class into available periods, one only has to specify the code of the class in addition to the command. It is not necessary to specify every assignment and period.

Although the use of the set-defining system allows a great variety of operations within one command, there is one important multiple load/unload operation for which the above type of command cannot

be used directly. This is period exchanging, in which all assignments in one period are exchanged with all those in another. This operation is used frequently as a simple means of satisfying period or distribution constraints. More complex operations of a similar type are also carried out, such as exchanging pairs or sets of periods and a rotation between three or more periods. Exchanging periods is tedious when done assignment by assignment with the basic commands and a special command is provided in the current system for this purpose.

4.4.2 Changing Requirements

One of the virtues of interaction is that it allows requirements to be altered during construction. Ideally, we wish to be able to change requirements as easily as the manual timetabler can change tickets within blocks. The command to change requirements must be quick and easy to use, but the variety of possible different types of change makes this difficult to achieve.

The basic requirement changes are as follows:

a) Add a teacher (with subject), class or classroom to a requirement.
b) Delete a teacher, class or classroom from a requirement.
c) Change a numeric value associated with a requirement (e.g. total number of single periods required).
d) Allow or disallow a period for a requirement.
e) Alter a distribution constraint for a requirement.

The provision of composite operations is again important here. The requirement changing command allows a set of changes to be made to all members of a set of requirements defined by the set-defining routine (section 4.3.7). Thus it is possible, for example, to forbid a period for all requirements involving a given teacher, without having to specify the individual requirements. However, specialised composite operations still have to

be provided for certain frequently made alterations, such as splitting or tying blocks. Even a simple split or tie involves a fairly complex series of elementary changes.

Associated with the requirement changing operation is a display which allows the timetabler to observe the effects of requirement changing. This display gives the details for all requirements in the list or for a set of requirements as defined by the set-defining system.

Commands are also provided for establishing and altering lists of teacher, class, classroom and subject codes and other similar data. These operations also have associated displays which permit the timetabler to check the effects of alterations.

4.5 Conclusions on Interaction Design

The experience with display designing has shown the difficulties in providing an effective man-machine interface. Some of the displays tested appear to provide all or most of the information that might be needed, but in fact turn out to be useless in practice. In particular, the major problem areas are:

a) Deciding what information is needed by the timetabler. Manual timetabling is a complex and poorly understood activity and considerable variation in technique is apparent. There is therefore no firm base on which to make decisions in this area. One can rely only on manual strategies which are commonly used and are relatively well defined.

b) Formatting the displays. The display format determines how quickly the timetabler interprets the information displayed. If the timetabler is not able to obtain the information within a time of the order of a few seconds, he will tend to avoid using the display. Correct formatting is thus vital to the success of the display, but there is no simple logical relationship between the format and the interpretation time.

The visual scanning and searching processes adopted by the timetabler determine the interpretation time, but even the basic perceptual processes involved are poorly understood.

However a few general conclusions may be drawn from the experience

1) A display of information for one object only has two advantages over a display which attempts to give all information relating to the problem without emphasis on any particular object:

   a) It gives a clearer representation of that information, by simplifying the formatting problem.
   b) It permits the use of small display screens which are inexpensive and available at the present time.

2) Many tasks appear to require information for several objects, usually about 2 or 3. The display should, if possible, allow all that information to be present simultaneously it it is required.

3) Formats which permit the use of simple linear scanning are to be preferred over formats for which complex visual searching is required.

4) Objects appearing in a complex display must be represented by 'names'. These names must be:

   a) unambiguous.
   b) compact, so as to minimise the size and complexity of the overall display.
   c) able to be used for specifying the objects to the computer quickly via commands.
   d) easy to learn, to remember, and to associate with the objects themselves.

On the 'man-to-machine' side, the main conclusion that can be drawn is that time and tedium is saved by combining a sequence of operations having a common purpose into a single operation

with a simple command. The set defining system in particular allows this to be done for several operations and for some of the displays from one central point in the timetabling program.

4.6 The Theory of Newell and Simon

An alternative view of the problem of man-machine interaction in timetabling is taken in this section. The theory of human problem-solving behaviour as proposed by Newell and Simon (1972) is applied to interactive timetabling, to give some insight into the effect of human limitations on the design of the computer aid. Newell and Simon proposed their theory after extensive studies of protocols of three different types of problem cryptarithmetic, logic and chess. The general strategies employed by the human in performing these tasks appear to have considerable similarity to those applied in timetabling. The most directly relevant portions of the theory are described and discussed below.

4.6.1 Fundamental Characteristics of the Human IPS

Certain characteristics of the human information processing system (IPS) appear to be invariant over all tasks. In particular there are restrictions on memory and processing time which determi to some extent, the style of problem-solving procedure adopted by the human. One of the functions of an interactive aid should be to act as an extension to the human problem-solver in order to reduce the effect of these restrictions.

4.6.1.1 Long Term Memory

Human long term memory (LTM) can be considered effectively infinit in capacity and potentially infinite in vocabulary of symbols. It is usually described as being associative. Associativity can be considered as the designation of stored symbol structures by symbols drawn from the potential vocabulary. The time to read a symbol structure from associative memory is of the order of a few hundred milliseconds. This time does not increase appreciably

with the size of the retrieval ensemble. Thus, access to LTM must be parallel to some extent.

The main restriction on LTM which limits its use in problem-solving is the 'write' time. To store a symbolized internal representation of a stimulus containing N familiar subpatterns or 'chunks' takes about 5N or 10N seconds. Thus it cannot be used for storing transient information such as a particular state of the timetable.

The main function of LTM in timetabling, as well as in storing procedures, appears to be in storing details of experience gained with various requirements during the timetabling process. Such experience includes how 'difficult' it is to move the assignments of a requirement around the timetable and what kind of requirement it is. The means used to access this experience is not definite in manual timetabling, but in the computer-aided timetabling it is, of course, the two character requirement codes.

4.6.1.2 Short Term Memory (STM)

STM has a very small capacity of up to about seven or so symbols or 'chunks'. However, each of these may point to a structure of arbitrary size and complexity in LTM. Access time is very short, both for reading and writing. For problem-solving the STM can be considered as consisting of the set of symbols that are immediately available to the IPS at a given instant of time.

The contents of STM have a tendency to decay. The rehearsal that is therefore necessary slows down the problem-solving process and errors are likely to result. Thus, an important benefit of the interactive aid is to reduce the demand on STM, by reducing the need for the timetabler to remember transient information.

4.6.1.3 External Memory (EM)

External memory constitutes the paper and pencil in cryptarithmeti and logic, the chessboard in chess, and the timetabling board in manual timetabling. The portion of EM in foveal view is almost

instantaneously accessible and can be considered as an extension of STM.

Access to other areas of EM require saccadic processes indexed by information in STM, or even visual scanning. It takes a few hundred milliseconds either to read from a fixated domain to STM or to perform a saccade to another point in the visual field. Under some circumstances a larger region of EM than just the foveal view may be included in the extension of STM.

Thus, the nature of the EM has a direct influence on problem-solving behaviour. EM in the case of computer-aided timetabling consists only of the current display, since access time to any other information stored in the computer is much greater than a few hundred milliseconds. Ideally, the current display should have all and only the information required for a particular subtask available simultaneously.

Furthermore, the displays must be designed to assist in the process of accessing areas not in the foveal view. Peripheral vision may play a part here and clues to location must therefore be easily visible. For example, we must have the lines between the days to assist in locating periods. The arrangement of displa symbols in neat rows and columns also assists in location, since the rows and columns themselves provide positional clues. The lack of such positional clues probably contributed to the failure of the display in Figure 4.3. The regular patterns formed by the repeated characters forming the blocks in figure 4.4 are readily perceived and act as positional clues.

4.6.1.4 Elementary Processes

All processes are considered to take their inputs from STM and to leave the results in STM. The human IPS is basically a serial system which can do only one elementary information process at a time. Elementary processes in STM take of the order

of 40 milliseconds or so and consist of simple compare and replace operations (e.g. comparing a ticket on the board with a target ticket in manual clash-checking). They each involve only one or two input and output symbols. More complex processes either involve access to LTM or consist of sequences of subprocesses.

In the design of a timetabling aid we wish to reduce the number of those human elementary information processes involved in carrying out a task in timetabling which can be considered to be fundamental; for example, determining whether or not a period is free, or whether a certain requirement clashes in some period are probably fundamental processes. Hence, the displays are designed to give this information in the most direct form possible, e.g., by means of special symbols for free periods and forbidden periods.

4.6.2 The Problem Space

Newell and Simon postulate that problem-solving takes place in a closed problem space. The problem space consists of a set of 'knowledge states', a set of operators which produce new knowledge states from existing knowledge states, an initial state of knowledge which the problem solver has at the start, and a set of final desired states.

The current knowledge state is the content of STM together with a small quantity of information in LTM, plus a variable amount in EM. The immediate knowledge state is small, but an extended knowledge state can be defined which consists of the knowledge that can be accessed from EM or LTM via pointers in STM.

Also available to the problem solver is path information indicating how the state was reached and what other actions were taken in this state if it has been previously visited, access information to other knowledge states previously reached, and reference information in LTM or EM that is constant over the course of problem-solving.

Certain features of problem spaces used by humans appear to be invariant over tasks and subjects:

a) The set of operators is small and finite.
b) The available set of alternative nodes in the problem space to which the problem solver may return is limited to about one or two nodes.
c) The residence time in each particular knowledge state before generation of the next state is of the order of seconds.
d) The moves from one state to the next are mostly incremental.
e) The search in the space involves backup - returning to old knowledge states and hence abandoning of knowledge state information.
f) The knowledge state is typically only moderate in size, containing a few dozen symbols.

A typical mode of operation of the interactive timetabling aid tends to support these hypotheses. This mode consists of the following time-sequence of operations:

a) Generate a display on the screen immediately below the previously generated displays.
b) Examine this display to gain new information. By doing this the timetabler moves to a new knowledge state.
c) Choose a possible likely assignment move, or some other change to the timetable (an operator).
d) Determine the consequences of this change. To do this may require a new display. Return to (a).

The time spent in examining each display is of the order of seconds. The time required to generate each display should be at the most of the same order of magnitude.

An advantage of having several displays on the screen at once is that it provides information on earlier states to which the timetabler can back up if necessary. In this way it increases the number of backup states available to the problem-solver. However, a disadvantage of the storage-tube display is the impossibility of erasing displays generated below the chosen backup state. Instead, time must be spent on erasing the whole screen and regenerating all displays up to the one representing the backup state.

4.6.3 Concluding Remarks On Newell & Simon Theory

Newell and Simon's theory generally agrees with our empirical findings in the restricted context of interactive school time-tabling. Although it sheds some light on the man-machine aspects of this area, the theory is still inadequate as a practical basis on which to design displays and on which to assess the overall efficacy of interaction.

CHAPTER 5

TIMETABLE INFEASIBILITY TESTING

5.1 Introduction

The displays and timetabling operations so far described provide the timetabler with the ability to observe the state of the timetable and to make changes quickly, while taking advantage of the computer's capability for data storage, manipulation and selection.

Nevertheless manual timetabling is characterised by 'difficult' stages, in which much time is spent manipulating assignments but little progress is made towards a solution in terms of the number of assignments so far fitted.

These difficult stages will still appear in computer-aided timetabling unless some suitable means for dealing with these stages is provided in addition to the displays and operations. This chapter and the next consider two different approaches to the problem of difficult stages.

The earlier in the construction process that a possible difficulty is detected, the easier it is to make a correction which will avoid or reduce the difficulty. This is because it is easier to move assignments in a less full timetable, and because the search for a possible cause of the difficulty can be narrowed down to a smaller subset of the assignments. This chapter presents a comprehensive infeasibility testing method which will detect at an early stage that a timetable cannot be completed. Two practical infeasibility tests suitable for interactive use are also described.

The main infeasibility testing method presented in section 5.2 is based on the timetabling algorithm proposed by Gotlieb (1963) and modified by Lions (1966a). The important feature of this algorithm is the use of 'availability reduction' to eliminate choices that can be shown to lead to infeasibility. The method is described in the context of the practical timetabling problem

defined in section 2.4. Section 5.2 ends by discussing the constraints imposed by the interactive environment on feasibility testing and by presenting a 'compatibility check' suitable for use in interactive timetabling.

In section 5.3 it is shown that the general method can be extended by invoking the concept of mutually clashing sets. The second infeasibility test suitable for practical use is based on the extended method and is presented in this section.

5.2 The Availability Reduction Method

This method was originally proposed by Gotlieb (1963) and was modified by Lions (1966a). The basis of the method is a theorem due to Hall (1935) on the existence of a system of distinct representatives.

From a partial timetable an 'availability matrix' can be defined and special procedures exist to 'reduce' the elements of this matrix from 1 to 0 in such a way that the resulting reduced matrix contains all possible solutions to the problem of completing the original partial timetable. This reduced matrix is used to decide which entry is to be made into the timetable next. The entry is made, a new availability matrix is defined and reduced, and a further choice is made, and so on until the timetable is complete.

The reduction of the availability matrix reduces the choice of possible entries in such a way as to increase the chance of finding a solution if one exists.

The following definitions use the terminology introduced in section 2.4.

5.2.1 The Availability Matrix

Definition 5.1. Given a partial timetable T', the free period matrix is a zero-one matrix $A = [A_{rp}]$ defined by:

$A_{rp} = 1$ if p is free for r, i.e.

$$X_{rk} + \Sigma X_{ik} T'_{ip} \leq L_k, \quad \forall k$$

$A_{rp} = 0$ if p is not free for r.

Definition 5.2. Given a partial timetable T', an availability matrix for T' is any zero-one matrix $B = [B_{rp}]$ whose elements satisfy the condition:
For all complete timetables T which contain T' (i.e. $T_{rp} \geq T'_{rp}$), $0 \leq T_{rp} - T'_{rp} \leq B_{rp} \leq A_{rp}$ $\forall r,p$
where $A = [A_{rp}]$ is the free period matrix for T'.

Thus an availability matrix contains all solutions that can be obtained with T' as the starting point, and is itself contained in the free period matrix.

5.2.2 The Availability Reduction Principle

Suppose that a partial timetable T' has an associated availability matrix B. Suppose that for some requirement r_o and some period p_o,

$$T'_{r_o p_o} = 0 \quad \text{and} \quad B_{r_o p_o} = 1$$

but $T_{r_o p_o} = 0$ in all timetables T containing T'. In other words, p_o is free for r_o, but no complete timetable T containing T' has r_o allocated to p_o. Then $B_{r_o p_o}$ can be changed to 0 and the resulting matrix B' still satisfies the condition in definition 5.2. It still contains all solutions that can be obtained with T' as the starting point. Hence B' is an availability matrix for T' which has fewer non-zero elements than B. B' is a 'better' matrix than B in the sense that it is closer to the minimal availability matrix defined by

$$B_{min} = \bigcup_{T} (T - T').$$

$B_{r_o p_o}$ is called an infeasible element of B.

The object of the procedure of availability reduction is to find and reduce to 0 as many infeasible elements in B as possible. Since during timetable construction it is not known what complete timetables T containing T' exist, methods for detecting infeasible elements must be heuristic to some degree.

5.2.3 Schedules for Teachers and Classes

In the Gotlieb method as modified by Lions, infeasible elements are found by attempting to construct 'subtimetables' or 'schedules' for teachers, classes and periods. The method was originally proposed for the simple timetable problem in which every requirement consisted of exactly one teacher and exactly one class. Teacher and class schedule construction can be extended directly to the more general problem defined in section 2.4. The generalisation of period schedule construction is less direct and will be discussed in section 5.2.7.

Definition 5.3. Given a partial timetable T' and an associated availability matrix B, a schedule for a teacher or class (or any single-life item) k is a zero-one matrix $U = [U_{ip}]$ for which

a) $\forall i, \quad \sum_p U_{ip} = N_{r_i} - \sum_p T'_{r_i p} \triangleq M_i$

where $\{r_1, r_2, \ldots, r_n\} = \{r_i : X_{rk} = 1\}$,

i.e. the number of elements in a row i of U must correspond to the number of assignments of r_i remaining to be entered into T';

b) $\forall \ell, p, \quad \sum_i X_{r_i \ell} U_{ip} \leqslant L_\ell - \sum_r X_{r\ell} T'_{rp}$,

i.e. the element U_{ip} must not 'clash' with any other element of U or any assignment in T';

c) $\forall i, p, \quad U_{ip} \leqslant B_{r_i p}$,

i.e. the matrix U must be contained in the appropriate section of the availability matrix B.

Condition (b) can be replaced by the simpler condition

b') $\forall p, \quad \sum_i U_{ip} \leqslant 1$,

since this implies that only one of $U_{1p}, U_{2p}, \ldots, U_{i_{max}p}$ will be 1. Hence, if $U_{jp} = 1$,

$$\sum_i X_{r_i \ell} U_{ip} = X_{r_j \ell} .$$

From condition (c) of definition 5.3, $B_{r_jp} = 1$, i.e. period p is free for r_j. The condition in definition 2.13 therefore applies:

$$X_{r_j\ell} \leqslant L_\ell - \sum_r X_{r\ell}T'_{rp} \qquad \forall\ell$$

which can be rewritten as:

$$\sum_i X_{r_i\ell}U_{ip} \leqslant L_\ell - \sum_r X_{r\ell}T'_{rp} \qquad \forall\ell.$$

Hence the three conditions are now:

(a) $\forall i, \sum_p U_{ip} = M_i$, (b') $\forall p, \sum_i U_{ip} \leqslant 1$, (c) $\forall i,p, U_{ip} \leqslant B_{r_ip}$,

so that the only connections with the original timetabling problem are the values M_i and the availability matrix B.

In the following we shall represent a schedule in a format in which the elements of B are shown as 1's and 0's and the non-zero elements of U_{ip} are shown as x's (figure 5.1).

i								M_i
	1	1	0	0	0	0	x	1
	2	1	1	x	0	0	1	1
	3	x	0	0	1	0	1	1
	4	1	0	1	0	x	0	1
	5	1	0	0	x	0	0	1
	6	0	x	0	0	0	1	1

Figure 5.1
Schedule

5.2.4 Availability Reduction by Schedule Construction

From any complete timetable T we can extract a schedule U for any item k simply by putting $U_{ip} = T_{r_ip}$ (r_i defined as before). Hence, if we find in the partial timetable T' that there is an item for which no schedule can be constructed using the availability matrix B, then T' is infeasible. If schedules can be constructed for all items, but no schedule U in which $U_{ip} = 1$ for some given i and p can be constructed for some item k, then the element B_{r_ip} is infeasible. This is because if no schedule exists in which $U_{ip} = 1$, then no complete time-

table can exist with $T_{r_{ip}} = 1$. Hence we can reduce availabilities by applying the following procedure to each $B_{r_{ip}} = 1$:

a) Set $U_{ip} = 1$

b) Attempt to complete a schedule for item k in which $U_{ip} = 1$

c) If the schedule cannot be completed, reduce $B_{r_{ip}}$ to 0.

5.2.5 The Hungarian Algorithm (Lions (1966a) Kuhn (1955))

Availabilities are reduced in the Gotlieb - Lions method by the application of the Hungarian algorithm due to Kuhn (1955) to the problem of schedule construction. The major component of this algorithm is a subroutine called EXPAND which adds one new element to an incomplete schedule, rearranging elements already in the schedule if necessary. The algorithm is applied iteratively to an initially blank schedule until condition (a) of definition 5.3 is satisfied for all i. If at any stage EXPAND fails to generate a schedule with a new element, then the original problem has no solution.

EXPAND is essentially a breadth-first tree search which attempts to find a rearrangement of the 'x's in the incomplete schedule U to allow a new 'x' to be put into a specified row of U. It uses two auxiliary arrays R_p and S_p whose functions can be interpreted as follows:

R_p = Row-number of element of U which is moved to column p.

S_p = Source column of element moved to column p.

To minimise searching, the matrix U is expressed as a one-dimensional array:

$$Z_p = i \quad \text{iff} \quad U_{ip} = 1$$
$$= 0 \quad \text{iff} \quad U_{ip} = 0 \quad \forall i$$

This representation is unambiguous since not more than one non-zero element can appear in any one column of U by condition (b').

Two queues, Q_j and Q'_j, are used to keep track of rows searched by the algorithm.

5.2.5.1 Subroutine EXPAND

(i is initialised to the row into which the new element is to be added):

Initialise

1. $j_1 \leftarrow 1$, $j_2 \leftarrow 1$, $k \leftarrow 0$, $R_p \leftarrow 0$ for $p = 1$ to p_{max}.

Scan row i of schedule period by period

2. $p \leftarrow 1$
3. If $B_{r_i p} = 0$, go to 9. (Element not available)
4. If $R_p \neq 0$, go to 9. (Period already tested)
5. If $Z_p = 0$, go to 12. (No clashing in this period)
6. If $Z_p = i$, go to 9. (Clash with element in same row)

Save row number of clashing element in queue

7. $R_p \leftarrow i$, $S_p \leftarrow k$,
8. $Q_{j_1} \leftarrow p$, $Q'_{j_1} \leftarrow X_p$, $j_1 \leftarrow j_1 + 1$.
9. $p \leftarrow p + 1$, if $p < p_{max}$, go to 3.

Get next row number for searching from queue

10. If $j_1 = j_2$, then exit (no solution)
11. $k \leftarrow Q_{j_2}$, $i \leftarrow Q'_{j_2}$, $j_2 \leftarrow j_2 + 1$, go to 2.

Change schedule and enter new element

12. $Z_p \leftarrow i$
13. If $k = 0$, then exit (solution completed)
14. $p \leftarrow k$, $k \leftarrow S_p$, $i \leftarrow R_p$, go to 12.

5.2.5.2 Example of Operation of EXPAND

In this example it is required to enter an element into row 1 of the incomplete schedule in figure 5.2.

Figure 5.2

Incomplete Schedule

1	1	0	0	0	0	1	
2	1	x	1	0	0	1	
3	x	0	0	1	0	1	
4	1	0	1	0	x	0	
5	1	0	0	x	0	0	
6	0	1	0	0	0	x	
	3	2	0	5	4	6	X_p

The loop extending from step 3 to step 9 of the algorithm first searches row 1. The element in row 1 can go only into column 1 or column 6. Since in columns 1 and 6 there are schedule elements in rows 3 and 6 respectively, the row-numbers 3 and 6 are put onto the queue. After the search of row 1 is completed, row 3 is searched, being the next on the queue. Column 4 is the only untested column which has a non-zero element for row 3 and so row 5, the position of its solution element, is put on the queue. Row 6 is searched next, resulting in row 2 (from column 2) being put on the queue. The searching of row 5 produces no new entries on the queue. Finally searching row 2 reveals that column 3 has no schedule element, and hence a solution has been found. The portion of the algorithm from step 12 to step 14 changes the schedule and puts in the new element. Figure 5.3 shows the tree structure developed by the algorithm and the new schedule is shown in Figure 5.4.

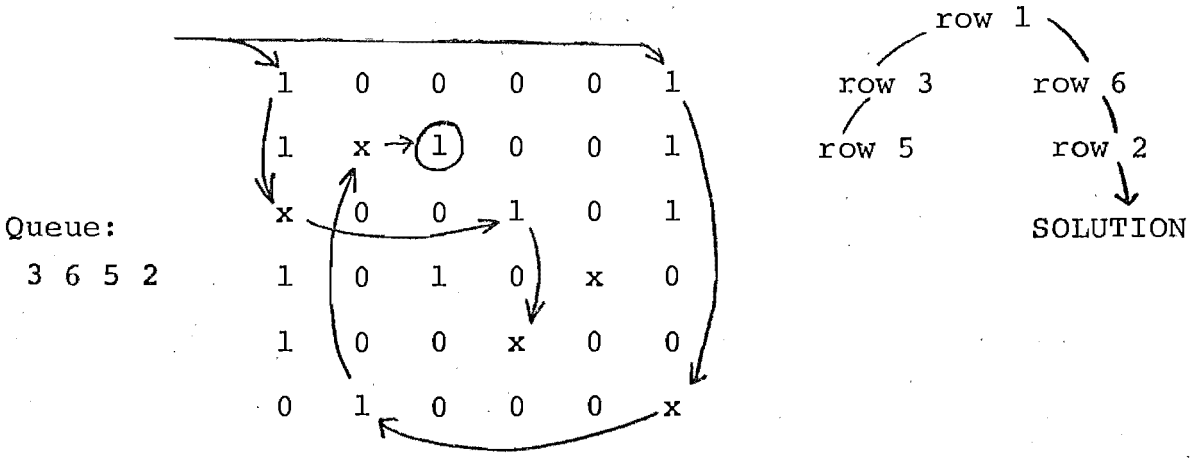


Figure 5.3 Tree Structure Developed by EXPAND

1	0	0	0	0	x
1	1	x	0	0	1
x	0	0	1	0	1
1	0	1	0	x	0
1	0	0	x	0	0
0	x	0	0	0	1

Figure 5.4 New Solution

5.2.5.3 Application of EXPAND

EXPAND is applied in two stages. The first stage is to apply EXPAND iteratively to construct a schedule for the teacher or class and to establish that the section of the timetable concerned is feasible. The second stage is to test each non-zero element of B in turn for feasibility.

To test the element B_{r_ip}, a non-zero element in row i of the schedule U is shifted to the position U_{ip} and 'fixed' to prevent EXPAND from attempting to move it elsewhere. (This can be done by setting R_p to i and by modifying EXPAND so that it does not clear the R array at the beginning). This action will result in at most one member of the schedule being displaced. Hence only one application of EXPAND is necessary either to restore the schedule or to demonstrate the infeasibility of the element B_{r_ip}. In fact, EXPAND will establish the feasibility or infeasibility of more than one element on one application as will be shown in the next section.

5.2.6 Minimising The Use Of EXPAND (Lions 1966a).

5.2.6.1 The Hall Condition for the Existence of a Schedule

The following condition is necessary for the existence of a schedule for a given availability matrix B_{r_ip} and for a given array of M_i's.

Hall (1935) proved a theorem on the existence of a system of distinct representatives for a collection of subsets of a set. The theorem can be directly interpreted as stating that the following condition is also sufficient.

Condition 5.4. For any subset S of {1,2,...,n},

$$\sum_{p} \bigcup_{j \in S} B_{r_j p} \geq \sum_{j \in S} M_j$$

The number of non-zero bits in the union of any subset of the rows of B must be greater than or equal to the total number of 'x's that must be entered into this particular subset of the rows.

5.2.6.2 Tight Sets

If the stronger condition $\sum_{p} \bigcup_{j \in S} B_{r_j p} = \sum_{j \in S} M_j$ holds for some subset S of the rows of B, then it is possible to show that a number of elements of B are infeasible.

Definition 5.5. A tight set is a subset S of {1,2,...,n} for which

$$\sum_{p} \bigcup_{j \in S} B_{r_i p} = \sum_{j \in S} M_j .$$

Definition 5.6. A proper tight set is a tight set which contains no tight subset.

		p							
		1	2	3	4	5	6	7	M_i
i	1	1	1	0	1	0	1	1	1
	2	1	1	1	0	1	0	1	2
	3	0	0	0	1	0	1	1	2
	4	0	0	0	1	1	1	0	2

Proper Tight Set

Figure 5.5 Example of Proper Tight Set

In figure 5.5, the union of rows 3 and 4 is 0001111 and the number of 1's in this union is 4. The sum of M_3 and M_4 is also 4. Hence rows 3 and 4 form a tight set, which is also a proper tight set since neither row 3 nor row 4 satisfy the tight-set condition.

Corresponding to the tight set S, there is a set of periods P_S defined by:

$$P_S = \{p : \bigcup_{j \in S} B_{r_j p} = 1\}.$$

Consider now the element B_{r_tq}, where $t \notin S$ and $q \in P_S$. If this element is made part of the schedule, then B_{r_iq} becomes zero for all i because all other requirements r_i have the same item k as r_t and hence clash with r_t. Hence

$$\bigcup_{i=1}^{n} B_{r_iq} = 0$$

and in particular

$$\bigcup_{j \in S} B_{r_jq} = 0$$

after this entry is made. Since $q \in P_S$, $\bigcup_{j \in S} B_{r_jq}$ was originally equal to 1. Thus we reduced the sum $\sum_p \bigcup_{j \in S} B_{r_jp}$ by 1 and so now

$$\sum_p \bigcup_{j \in S} B_{r_jp} < \sum_{j \in S} M_j,$$

violating condition 5.4. All non-zero elements B_{r_tq}, $t \in S$, $q \in P_S$, are infeasible for this reason. These elements are underlined in the example of figure 5.5.

5.2.6.3 Use of EXPAND to Detect Tight Sets

Suppose that EXPAND is used to test an infeasible element B_{r_tq}. The operation of EXPAND is to search the row j containing the schedule element U_{jq}, and from this row it will access and search other rows of a tight set S containing row j. When it finds that it cannot insert a new element into the period set P_S, it will terminate. The rows of the tight set will be available in the queue Q'. Other elements of B in P_S but not in S can be reduced to zero as described above.

	1	2	3	4	5	6	7	8	9
1	x	0	1	0	1	0	0	0	0
2	1	x	0	0	1	0	1	1	1
3	1	1	x	1	0	0	1	0	1
4	0	0	0	x	1	1	1	1	0
5	0	0	0	1	x	0	1	0	1
6	0	0	0	0	0	x	1	1	0
7	0	0	0	1	0	1	x	1	1
8	0	0	0	0	0	0	0	x	1
9	0	0	0	0	0	0	0	1	x

Figure 5.6

Detection of Tight Sets

In the example of figure 5.6 the elements of the schedule lie along the main diagonal. The non-zero elements in the shaded area are all infeasible. The set {8,9} is a proper tight set. The sets {4,5,6,7} and {1,2,3} become proper tight sets if infeasible elements are reduced to zero.

The operation of EXPAND in testing B(1,5) would be to search rows 5,4,7,9,6 and 8 before terminating. The rows {5,4,7,9,6,8} form a tight set consisting of the proper tight sets {4,5,6,7} and {8,9}. This single application of EXPAND would thus reveal the infeasibility of all the underlined elements.

5.2.6.4 Confirmation of More Than One Element

Each successful application of EXPAND in the testing of the feasibility of elements results in a new schedule being formed. This new schedule has several elements different from the old one. Therefore computation can be reduced by marking all of these new elements as being feasible and not testing them again.

Computation can be reduced still further in the following way. Consider the schedule of figure 5.7(a). The application of EXPAND in the testing of B(1,2) results in the new schedule of figure 5.7(b). By connecting together old and new schedule elements alternately with horizontal and vertical lines, we form a closed loop (figure 5.7(c)). This loop defines a set of columns and a set of rows. Any non-zero element at the intersection of one of these rows and one of these columns can be labelled as feasible and not tested again. This is because a new loop can be formed which includes only elements in the original loop together with the element under consideration. Figure 5.8 shows new loops for the elements B(4,2) and B(1,4).

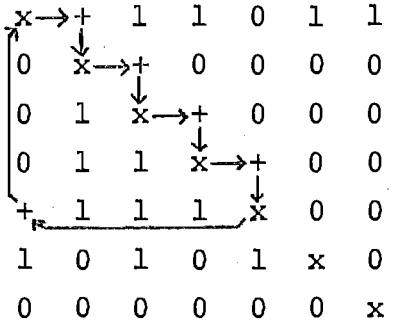
One application of EXPAND can thus demonstrate the feasibility as well as the infeasibility of more than one element.

x	1	1	1	0	1	1
0	x	1	0	0	0	0
0	1	x	1	0	0	0
0	1	1	x	1	0	0
1	1	1	1	x	0	0
1	0	1	0	1	x	0
0	0	0	0	0	0	x

(a) Original Schedule

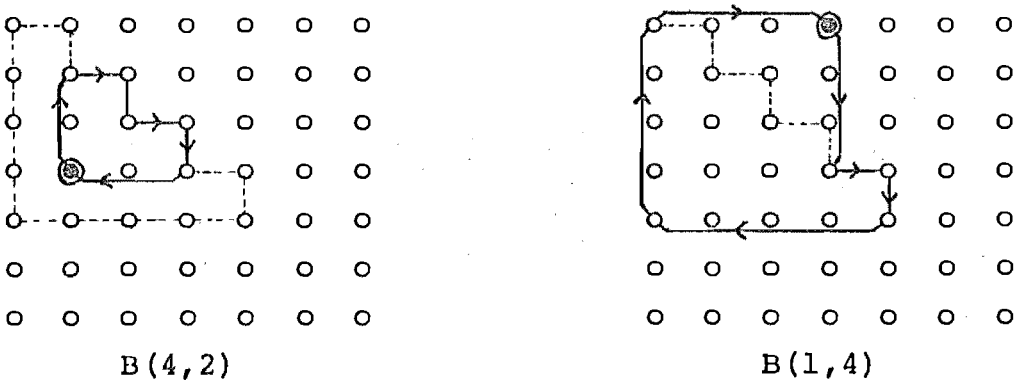
1	x	1	1	0	1	1
0	1	x	0	0	0	0
0	1	1	x	0	0	0
0	1	1	1	x	0	0
x	1	1	1	1	0	0
1	0	1	0	1	x	0
0	0	0	0	0	0	x

(b) New Schedule



(c) Connection of Old and New Elements

Figure 5.7 Confirmation of More than One Element



B(4,2) B(1,4)

Figure 5.8 Connection Loops for B(4,2) and B(1,4)

The algorithm EXPAND and the above examples were taken directly from Lions (1966a), with only slight modifications. The method is directly applicable to teacher and class schedules in the timetabling problem involving blocks. In the following sections we consider the task of attempting to construct schedules for periods in this more complex type of timetabling problem.

5.2.7 The Problem of Period Schedules

Gotlieb (1963) proposed the availability reduction method for the simplified problem in which each requirement consists of exactly one teacher and exactly one class. The availabilities for this problem can be expressed as a 3-dimensional array whose axes represent teachers, classes and periods, respectively. The value of the element B_{tcp} of this array is 1 or 0 according to whether teacher t is, or is not, free to take class c during period p.

Availability reduction is carried out by finding and reducing infeasible elements in the following 2-dimensional sections of this array - (Figure 5.9).

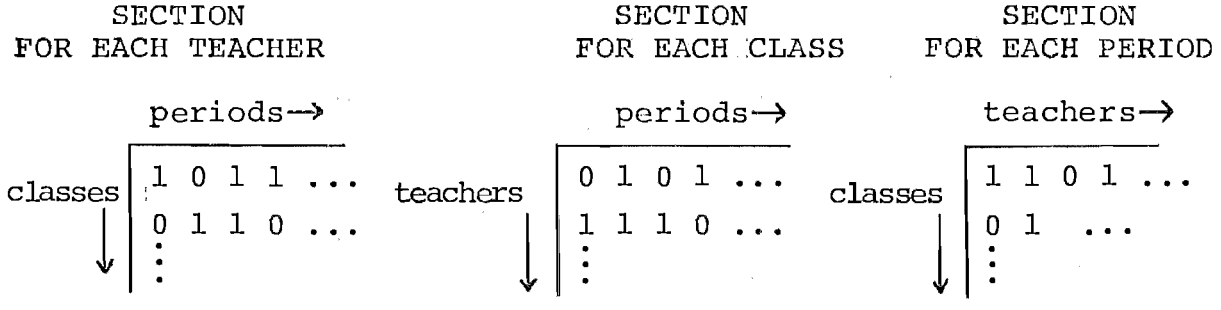


Figure 5.9 2-Dimensional Sections Of The Availability Array

This can be done, either by searching for tight sets (Gotlieb, 1963) or by using the Hungarian method (Lions,1966a).

In the more general problem in which each requirement can have any number of teachers and any number of classes, the previous sections showed that we can apply availability reduction to arrays whose rows correspond to requirements involving a given teacher or class

and whose columns correspond to periods. These arrays are directly equivalent to the class-period and the teacher-period sections in the simplified problem. This is because each row of each class-period or teacher-period section corresponds to a teacher-class meeting, which is, of course, a requirement.

However, it is not so easy to generalise availability reduction on teacher-class sections of the 3-D array. Each element in a teacher-class section corresponds to a requirement and requirements in the more general problem cannot be so conveniently arranged in a 2-dimensional matrix. The problem of constructing a schedule for a period must be considered either as one of finding a mutually non-clashing subset of the set of available requirements, or as one of (Lions, 1968) 'partitioning a set of n objects (items) into subsets when only certain subsets are allowed'.

5.2.7.1 Application of the Hungarian Method to Period Schedules

The Hungarian algorithm is essentially a breadth-first tree search which attempts to construct a 'path' from the given unfilled row to an empty column of the availability array. A path consists of an alternating sequence of vertical and horizontal links. Each vertical link connects a '1' in the array to an element of the original schedule in the same column, and each horizontal link connects a schedule element to a '1' in the same row (Figure 5.3).

The Hungarian algorithm takes advantage of the special property that only minimal-length paths to any column of the array need be considered. Having found a path to any column, it makes an entry in the R_p array and does not test this column in any subsequent searching. Because the search is breadth-first, the first path it finds will always be the minimal one. This property results in a considerable reduction in the size of the search tree.

Unfortunately, this property does not hold for the more general period-schedule problem because the clash structure of the requirements is more complex and cannot be represented as a 2-dimensional

matrix. Nevertheless it is still possible to apply breadth-first tree-searching to the problem. The following section proposes a simple breadth-first tree search algorithm for period schedules. If applied to the simple class-teacher section problem, this algorithm would operate in a similar style to, but much less efficiently than, the previously described Hungarian algorithm.

5.2.7.2 Breadth-First Tree Search Algorithm for Period Schedules

For the purposes of this algorithm we define a class as a single-life item k for which

$$\sum_r N_r X_{rk} = p_{max},$$

i.e. which is involved in lessons in every period of the week. A complete schedule for a period must therefore have every class involved in some meeting. The incomplete schedule is presented to the algorithm in the form of a binary array U(r) which is defined by:

U(r) = 1 if requirement r is in the schedule
= 0 otherwise.

The nodes of the search tree are stored in a queue Q in the order that they are to be expanded. Associated with each element of the queue Q(q) is a pointer P(q) which points to the parent node of the node Q(q).

1. (Initialise queue pointers) $q_1 \leftarrow 0$, $q_2 \leftarrow 0$.
(Transfer initial schedule to auxiliary array) $V(r) \leftarrow U(r), \forall r$
(Clear fix flags) $F(r) \leftarrow 0, \forall r$.

2. (k_o becomes first class not involved in any meeting) Let k_o be the smallest k such that for each r, either $U(r)=0$ or $X_{rk}=0$. Let $R_{k_o} = \{r_1, r_2, r_3, \ldots r_n\} = \{r:X_{rk_o} = 1\}$
= set of all requirements that contain class k_o.

3. $i \leftarrow 1$

4. (Test availability) If $B_{r_i p} = 0$, go to 8.

5. (Test for clash with 'fixed' requirement) If $r_i * r$ and $F(r)=1$ for some r, go to 8.

6. (Put new branch on tree) $q_1 \leftarrow q_1+1, Q(q_1) \leftarrow r_i, P(q_1) \leftarrow q_2$

7. (Is this a solution?) If, for all k, either $X_{r_i k} = 1$, or there exists an r such that $U(r)=1$ and $X_{rk}=1$ and r_i does not clash with r, then exit (solution found).

8. (Advance to next requirement containing class k_o) $i \leftarrow i+1$, if $i \leq n$ go to 4.

9. (Restore original schedule and clear fix flags) $U(r) \leftarrow V(r)$, $F(r) \leftarrow 0$, $\forall r$.

10. (Empty queue?) If $q_1 = q_2$ exit (no solution).

11. (Advance to next node on tree) $q_2 \leftarrow q_2+1$, $k \leftarrow q_2$.

12. (Remove clashes from original schedule) For all r such that $r * Q(k)$, $U(r) \leftarrow 0$.

13. (Put requirement from tree into schedule and fix) $U(Q(k)) \leftarrow 1$, $F(Q(k)) \leftarrow 1$.

14. (Advance k to point to next requirement to be put in) $k \leftarrow P(k)$, if $k = 0$ go to 2, otherwise go to 12.

5.2.7.3 Example of Operation of Breadth-First Tree Search

Suppose we have five classes a,b,c,d,e, six teachers 1,2,3,4,5,6 and requirements as in figure 5.10.

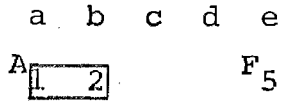
	a	b	c	d	e	1	2	3	4	5	6	
A	1	1	0	0	0	1	1	0	0	0	0	1
B	0	0	1	1	1	0	1	0	0	1	1	1
C	1	0	0	0	0	0	0	0	0	1	0	1
D	0	1	1	0	0	0	0	1	1	0	0	1
E	0	0	0	1	0	0	1	0	0	0	0	1
F	0	0	0	0	1	0	0	0	0	1	0	1
G	1	1	0	0	0	0	0	1	0	1	0	1
H	0	0	1	0	0	1	0	0	0	0	0	1
I	0	0	0	1	1	1	0	0	0	0	1	1
						X_{rk}						N_r

or

a	b	c	d	e
A [1 2]	B [2 5 6]			
C_5	$D_{[3\ 4]}$	E_2	F_5	
$G_{[3\ 5]}$	H_1	$I_{[1\ 6]}$		

Figure 5.10 Example Problem for Breadth-first Tree Search

The initial schedule of figure 5.11(a) is presented in the form shown in figure 5.11(b).



(a)

	A	B	C	D	E	F	G	H	I
U(r):	1	0	0	0	0	1	0	0	0

(b)

Figure 5.11 Initial Schedule

We shall assume that the availabilities for all requirements in the period under consideration are equal to 1. In the following description of the operation of the breadth-first tree search, letters in brackets refer to figure 5.12.

The first class not in any meeting is c.

$R_c = \{B,D,H\}$. Hence r_1=B, r_2=D, r_3=H.

At the beginning there are no fixed requirements, so step 5 has no effect. Step 6 puts the first node on the tree (a). This is not a solution, since the entry of B into the schedule displaces A and leaves classes a and b unassigned.

Hence we advance to D, which is also put on to the tree (b), followed by H (c).

This completes the testing of the elements of R_c so we now generate a new schedule. q_2 is first advanced to the next entry in the queue and k is set equal to q_2 (d). Clashes of B are removed from the original schedule, which becomes empty (e). B is entered into the schedule and is fixed (represented by * in (f)). Since the tree node B has a null pointer, there are no more entries to be made into the schedule.

The first unassigned class in this schedule is a and $R_a = \{A,C,G\}$. All the elements of R_a clash with the fixed entry B in the schedule so no new nodes are created.

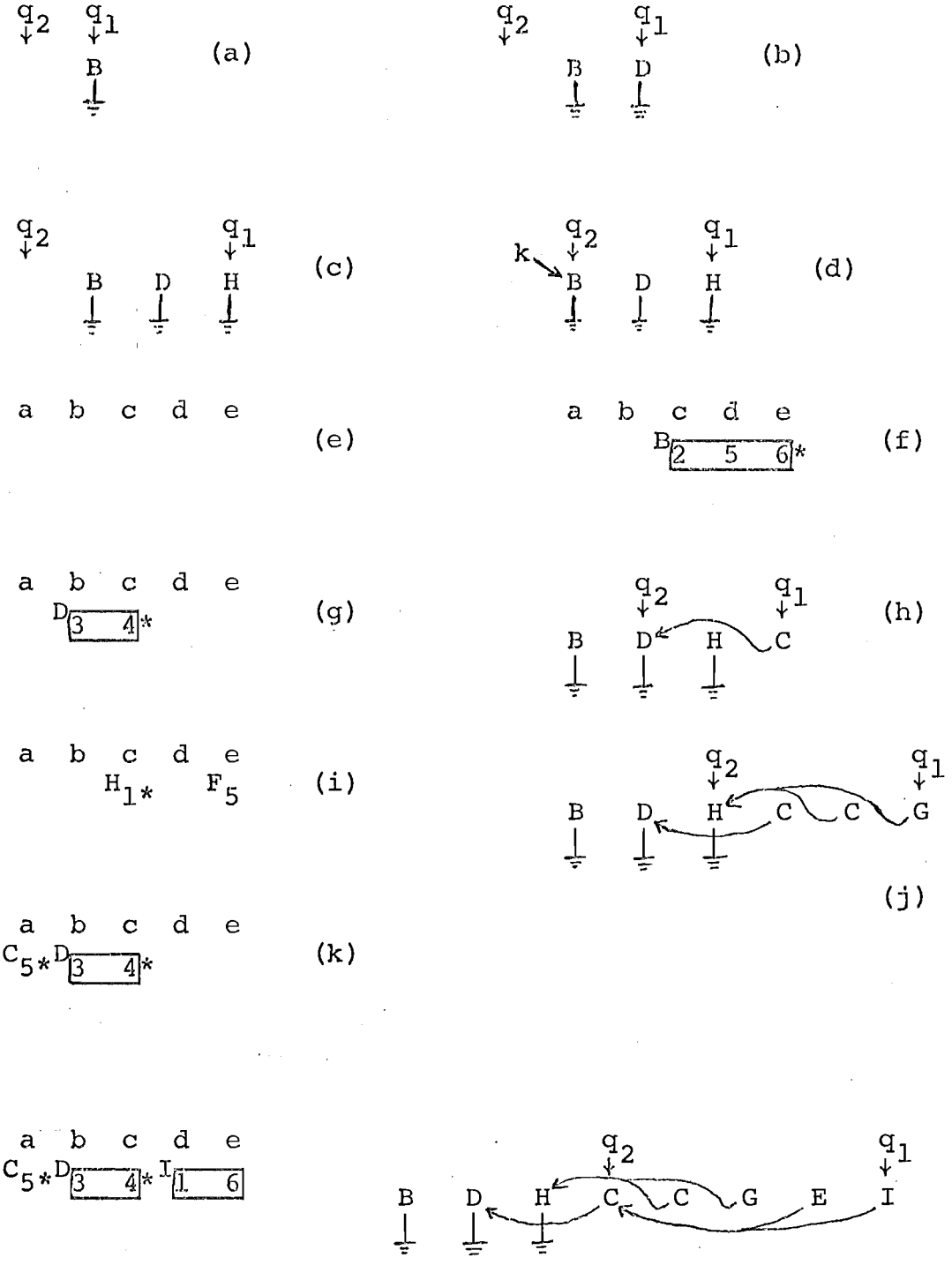


Figure 5.12 Execution of Breadth-first Tree Search

The original schedule is restored, q_2 is advanced to node D, and a new schedule is generated (g). Again the first unassigned class is a. The first requirement of R_a that does not clash with D is C. Therefore a new node is generated (h). Since the last element of R_a, G, clashes with D, q_2 is advanced to H giving the new schedule (i). The tree at this point is shown in (j).

q_2 is now advanced to the first tree node labelled C, and since this node is linked to an earlier node labelled D, both C and D are entered into the schedule and fixed (k).

The set R_d is {B,E,I}. B clashes with both C and D, E can be entered but leaves d unassigned, but the entry of I results in a solution (ℓ).

5.2.7.4 Application

The above period schedule algorithm is offered as a direct extension of the Hungarian algorithm to the general period schedule problem. An improvement to the efficiency could be made by using a simple heuristic to choose the next node of the tree to be expanded. An example of such a heuristic would be to choose the node representing the schedule with the least number of unassigned classes.

This algorithm, like the Hungarian algorithm, can be used for testing feasibility of the period or for reducing non-zero availabilities B_{rp} by the method of entering r into the schedule and fixing, then applying the algorithm to restore the schedule.

5.2.8 Availability Reduction in Timetabling

A timetabling program using availability reduction would proceed as follows:

1. Apply the availability reduction algorithms to teacher, class and period arrays repeatedly until no further elements of B are changed.

2. If, at any stage, an array becomes infeasible, either indicate failure or backtrack to an earlier position (if a record has been maintained of the construction procedure).

3. Otherwise choose any non-zero element B_{rp} in the reduced availability array and make the entry $T_{rp} \leftarrow 1$ into the timetable.

4. Change to zero the elements B_{sp} for all s for which s*r. Also change to zero all elements B_{rq} for q = 1 to p_{max} if $\sum_{p} T_{rp}$ is now equal to N_r.

5. If, for all r, $\sum_{p} T_{rp} = N_r$, the timetable is complete. Otherwise, go to 1.

To improve the performance, heuristics could be used in step 3. Such heuristics would attempt to minimise the chance of later infeasibility (by choosing an element in the smallest tight set found) and attempt to improve the acceptability of the timetable by the school (by favouring elements which give the best distribution of lessons through the week).

The efficiency of the program would be improved if unnecessary applications of availability reduction could be eliminated. For example, it is not necessary to reduce availabilities in an array which has remained unchanged since the last application of availability reduction. The addition of a meeting to the timetable results in only a limited number of elements of B being changed to zero. A considerable saving in computation would result if availability reduction were applied only to arrays containing changed elements (and thereafter to arrays containing newly reduced elements). Furthermore, a large proportion of these arrays have only one changed element. Lions (1971) describes several efficient procedures for reducing an array

which is derived from an existing reduced array by eliminating exactly one element. Application of these procedures where appropriate could result in a further significant reduction in computation.

5.2.9 Availability Reduction in the Interactive Environment

Availability reduction was proposed as a method for solving timetable problems. The intended mode of application was to start with an empty timetable, and to progressively add assignments until the timetable is complete or until infeasibility occurs. Distribution and other constraints characteristic of the practical problem had to be handled by separate heuristics, since the basic method does not handle such constraints directly.

As part of an interactive system, availability reduction appears to have value as a 'test' to detect that a partially constructed timetable can not be completed. It would be applied at a stage when every requirement as yet unallocated appears to have a large number of free periods, and it should detect infeasibility at an earlier stage than could any heuristic method. It should also indicate where to make the next allocations. However in fact the full availability method was found to be unsuitable for interactive use.

Execution times on the EAI 640 were of the order of 1-2 minutes for about 2 or 3 iterations through 30 classes and 60 teachers. (The test program did not attempt to construct period schedules). During this time either the timetable was found to be infeasible or only about 2 or 3 array elements were reduced. At this level of availability reduction performance, it was considered that the execution time was too great to allow effective interaction. It was decided that it is preferable to have a 'fast' check which can be repeatedly applied to test various arrangements in the timetable. Because in interactive timetabling assignments are frequently moved from one period to another, the reduced availability array resulting from the test did not remain applicable for long. The reason for the low number of array elements reduced is that inequality usually holds in condition 5.4 and availability

reduction occurs only when equality holds. Tight sets are thus relatively infrequent — usually either most sets are not tight or the timetable is infeasible.

Another shortcoming of the method in interactive timetabling is that it can be difficult for the timetabler to locate the cause of an infeasibility if several iterations of availability reduction were required. The cause may in fact be complex and spread widely throughout the timetable. On this basis the time spent in performing the extra iterations must be considered unjustified since the timetabler would have to spend further time in attempting to understand the cause for the one particular arrangement tested.

5.2.10 A Simplified Infeasibility Test

These disadvantages can be eliminated at the expense of thoroughness by the use of a simplified version of the Gotlieb-Lions method. The simplified method attempts to construct schedules for every teacher and class by means of the Hungarian algorithm, but does not reduce availabilities. Any teacher or class array sections that are found to be infeasible are reported to the timetabler, who may then concentrate his attention on the particular teacher or class. The algorithm for the simplified method is given in figure 5.13.

5.3 Mutually Clashing Sets in Availability Reduction

The availability reduction method considers sets of requirements which have a common teacher or class. The property of such sets that is exploited by the method is that every member clashes with every other member. The method can be readily extended to consider all maximal mutually clashing sets of requirements. Although a set of requirements with a common item is a mutually clashing set, it is not necessarily a maximal MCS. It is possible that a set of requirements may clash mutually yet have no common item; for example the set {A,B,C} where

requirement A has items a and b
B has items b and c
C has items c and a.

1. (1st item) $k \leftarrow 1$.
2. (Skip multiple-life items) If $L_k \neq 1$ go to 10.
3. Form array section $B_{r_{ip}}$ for item k.
4. (1st row of array section) $i \leftarrow 1$.
5. (Initialise counter of schedule elements in row i) $\ell \leftarrow 1$.
6. (Check whether number of schedule elements in row i has reached required number) If $\ell > N_{r_i}$ go to 9.
7. Use EXPAND to allocate a schedule element in row i of the array section. If EXPAND fails, report k to the timetabler and go to 10.
8. (Next schedule element to be allocated) $\ell \leftarrow \ell + 1$, go to 6.
9. (Row i contains required number of schedule elements — go to next row) $i \leftarrow i + 1$, if $i \leqslant n$ go to 5.
10. (Schedule completed for item k — get next item) $k \leftarrow k + 1$, if $k \leqslant$ total number of items go to 2 else halt.

Figure 5.13 Simplified Infeasibility Test

The extended availability reduction method attempts to construct schedules for every maximal MCS of requirements. Reduction of availabilities then proceeds in exactly the same manner as before. This generalised method will carry out a more comprehensive check but it requires an extra routine to locate maximal MCS's.

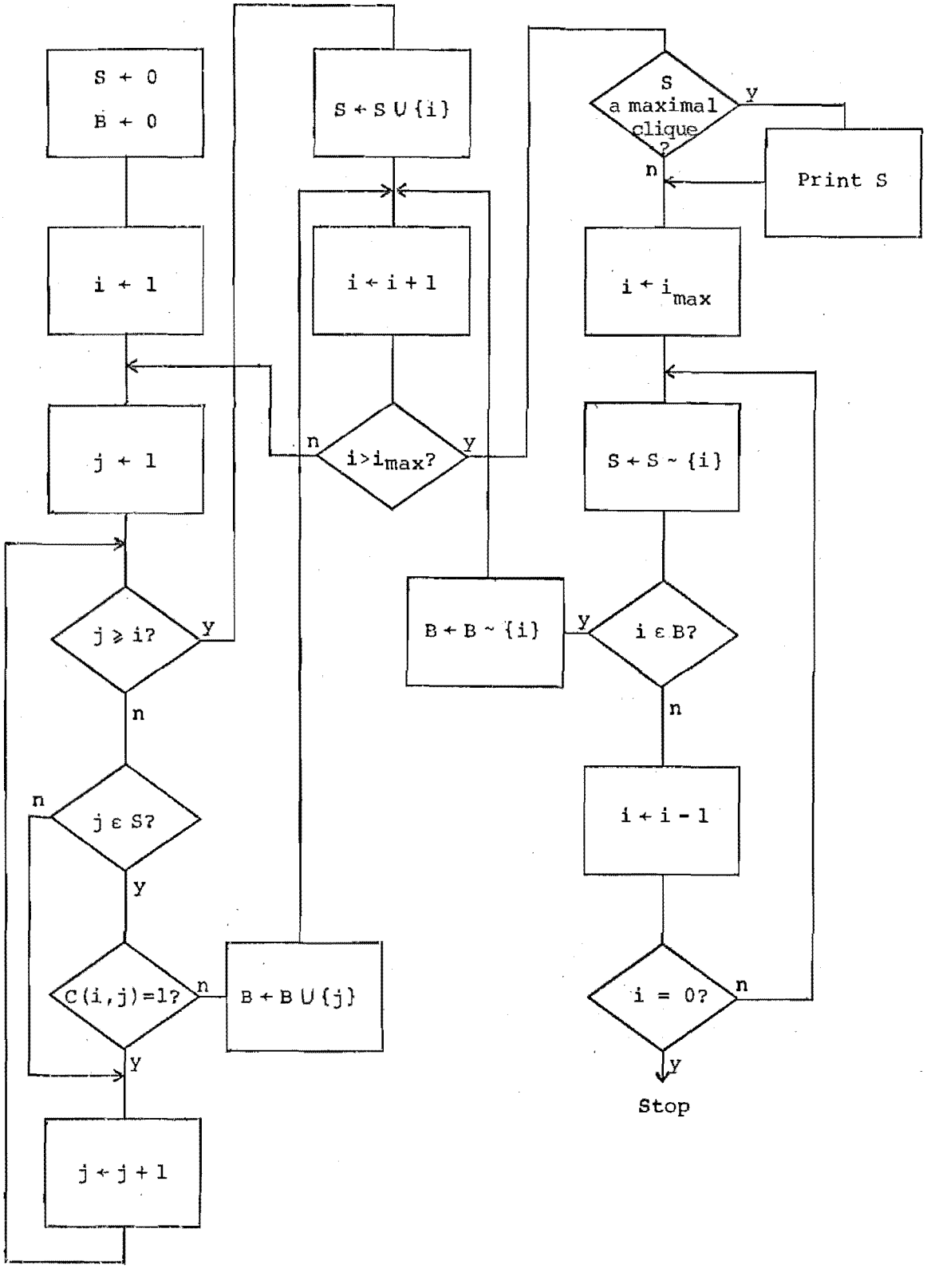
5.3.1 Location of Mutually Clashing Sets

Location of MMCS's is equivalent to searching for cliques in the clash graph. A clique is a maximal complete subgraph, i.e. a maximal subgraph in which each node is connected to every other node.

A flowchart of a simple clique-searching algorithm which requires minimal working storage is given in figure 5.14. The algorithm begins with the clique set S empty and the set used for backtracking, B, also empty. Elements are added to S in numerical order. An element 'i' is added to S only if it is found to clash with all elements 'j' already in S. For every other element 'i' the first element 'j' in S for which C(i,j)=0 is entered into the set B. This indicates that k must be

Figure 5.14 A Clique Searching Algorithm

S = clique set
B = backtracking set
C = clash matrix or clash-testing subroutine.



removed from S before i can be entered.

When all elements 'i' have been tested, S is printed out if it is found to be maximal. Then the latest additions to S are removed one by one until an element 'i' which is also in B is reached. This element is also removed from B and a new set of elements is added, starting from i+1, to form a new clique.

If extra storage is available, more sophisticated algorithms can be used to reduce the amount of computing required. A recursive algorithm whose behaviour is similar to that of the algorithm above, but which includes extra tests to short-cut unnecessary searching, is described by Cleary (1972). An efficient algorithm by Bierstone corrected by Mulligan and Corneil (1972) takes the different approach of building up stored cliques by adding elements one by one. Another applicable algorithm is that due to Das (1973).

5.3.2 Application in Practical Timetabling

As noted before, the full availability-reduction method was not considered suitable for interactive timetabling. If the method is extended to cover all MCS's a greater number of infeasible elements will be found but the computing overhead will increase even further. Experience suggests that simplification is therefore necessary.

A practical solution is to retain the MMCS searching algorithm and to use it before timetable construction is started to find any MCS that cannot be fitted into the available periods. As each MMCS is discovered by the searching algorithm, an attempt is made to construct a schedule for that MMCS into a timetable with all preassignments allocated. If this cannot be done, then the timetable problem is impossible and changes have to be made to the requirements. As in the previous timetable infeasibility test, the set of requirements nominated by the algorithm for examination by the timetabler is only a small and well-defined subset of the set of all requirements.

In fact if there are few preassignments and restricted periods, there is likely to be little advantage to be gained in carrying out a complete schedule-construction for each MMCS. A simple comparison between the total number of periods required by all members of the MMCS and the number of periods available in the timetable is all that should usually be necessary. At the start of timetable construction, most of the requirements have all periods available, and only a very limited number have more than a few periods unavailable. Thus subsets of MMCSs will generally be very 'loose', and will therefore rarely cause incompatibilities.

5.4 Conclusion

The two practical infeasibility tests presented in this chapter are a compromise between computing speed and ease of use on the one hand and thoroughness on the other. The full method has considerable shortcomings not only in terms of computation required, but also in terms of the utility of its results to the timetabler. On the other hand the simplified practical methods fill the need for effective tests which can be applied to initial or partially constructed timetables to detect and locate problems that would otherwise cause severe difficulties later on.

THE TREE SEARCH METHOD

6.1 Introduction

'Difficult' stages arise in manual timetabling in which the timetabler is forced to interchange a number of assignments on the board to allow a new assignment to be entered. Finding a non-clashing series of interchanges is time-consuming and often frustrating.

Although the infeasibility tests described in the last chapter reduce the incidence of such 'difficult' stages in computer-aided timetabling, they do not eliminate them entirely. At some point, assignments in a timetable being constructed with the aid of a computer will have to be moved to allow a new assignment to be entered. Even with the aid of computer displays giving direct information on clashes, the manual search required to find a satisfactory chain of interchanges can still be laborious.

This chapter shows that the computer can perform this task effectively and in a manner which enables the timetabler to retain full control. With the methods to be described, the timetabler can rectify difficult situations quickly and with a minimum of effort, even when human decisions such as changes to requirements and constrains become necessary.

The methods are based on a tree-searching algorithm proposed by Oliver (1968) as a means of constructing timetables by computer from single-teacher single-class requirements. Oliver proposed the tree search as an exhaustive method which will always find a timetable solution if one exists. The methods described here are more practical than Oliver's algorithm in that:

a) They can handle blocks and other complexities in the real school problem.

b) They are not designed to be exhaustive, but instead, they carry out a 'local' search from the current position. The search time for an exhaustive algorithm is prohibitively long for a practical problem.

6.2 Mode of Operation

The tree searching program forms another module in the interactive aid structure as depicted in Figure 4.1. The timetabler initiates execution by typing the command for tree searching followed by the code of the requirement to be entered, as well as (depending on the method used) a search depth parameter. The tree searching program will then search for a series of assignment interchanges or moves which enables the outstanding requirement to be entered without any other assignment being displaced out of the timetable. On finding a solution the program updates the timetable, reports the changes made, and returns control to the command routine. If a solution is not found the timetable is left in its original state.

6.3 Tree Searching Techniques

The search space of a variety of combinatorial problems besides school timetabling can be represented in the form of a tree. Tree structured problems arise in many areas of artificial intelligence, such as game playing (e.g. Samuel, 1969), theorem-proving (Gelernter et al 1959, 1960), and problem-solving (Ernst and Newell, 1969). A review of these areas is given by Slagle (1971). Tree structures also arise in various areas of operations research (e.g. Cherniavsky, 1972).

There is little in the methods used for searching the trees in these problem areas that can be directly transferred to the timetabling problem. The details of the methods used are tailored to the needs of the particular applications. However, the general approaches of depth-first and breadth-first searching of trees is common to most of these methods as well as to the algorithms described here. Recursive algorithms corresponding to these general approaches were given by Platts (1973). A detailed discussion of the back-tracking techniques required in depth-first searching is given by Golomb and Baumert (1965).

6.4 Basic Definitions

Before the algorithms themselves are introduced, a few concepts which are fundamental to the technique of tree searching will now be defined formally. The terminology used is introduced in section 2.4.

Definition 6.1. An interchange for a requirement r_o in a partial timetable T' is a matrix $I = [I_{rp}]$ ($I_{rp} = -1, 0, \text{or } 1$) for which

a) $T'' = T' + I_{rp}$ satisfies the no-clash condition (b) in definition 2.9, i.e.

$$\forall k,p, \quad \sum_r X_{rk} T''_{rp} \leqslant L_k$$

b) $$\sum_p I_{rp} = 0 \qquad r \neq r_o$$

$$\sum_p I_{r_o p} = 1$$

The effect of the interchange on the partial timetable T' is to add a new assignment for r_o. No other assignments are added or removed, but their positions may be changed. It is also necessary, of course, that

$$\sum_p T'_{r_o p} < N_{r_o}$$

since

$$\begin{aligned} \sum_p T''_{r_o p} &= \sum_p T'_{r_o p} + \sum_p I_{r_o p} \\ &= \sum_p T'_{r_o p} + 1 \\ &\leqslant N_{r_o} \end{aligned}$$

(by definition 2.11(a))

Definition 6.2. A chain in an interchange I is a sequence of of alternating requirements and periods $(r_1, p_1, r_2, p_2, \ldots, r_{n-1}, p_{n-1}, r_n)$ whose elements satisfy the properties

a) $I_{r_i p_i} = 1$ for i = 1 to n-1

b) $I_{r_{i+1} p_i} = -1$ for i = 1 to n-1

c) $r_i * r_{i+1}$ for i = 1 to n-1.

The elements of a chain are linked by clashing between the requirements r_i and r_{i+1} in period p_i, and by 'movement' of an assignment from p_i to p_{i+1} for each requirement r_{i+1}.

Definition 6.3. A tree-interchange I for a requirement r_o is an interchange for r_o in which, for every negative element $I_{sp} = -1$, there is a chain

$(r_1=r_o, p_1, r_2, \ldots, r_{n-1}, p_{n-1}=p, r_n=s)$.

Definition 6.4. A partial tree-interchange for a requirement r_o in T' is a matrix $I = [I_{rp}]$ ($I_{rp} = -1, 0,$ or 1) for which

a) $T'' = T' + I_{rp}$ satisfies condition (b) in definition 2.9

b) $\sum_p I_{r_o p} = 1$

$\sum_p I_{rp} \leq 0$ for $r \neq r_o$, i.e. assignments can be removed

c) For every negative element $I_{sp} = -1$, there exists a chain

$(r_1=r_o, p_1, r_2, \ldots, r_{n-1}, p_{n-1}=p, r_n=s)$.

Definition 6.5. An unsatisfied element in a partial tree-interchange I is a negative element $I_{rp} = -1$ in a row for which $\sum_p I_{rp} < 0$.

Definition 6.6. An unsatisfied requirement in I is a requirement r for which $\sum_p I_{rp} < 0$.

6.4.1 Examples

Suppose T' is the partial timetable of figure 6.1.

	1	2	3 periods
a	A_1	C_2	G_5
b	A_5	H_3	D_1
c	E_3	H_4	B_2
d		F_5	B_4

Figure 6.1
Partial Timetable T'

Teachers are numbered 1 to 5 and the classes are labelled a,b,c,d. The requirements are shown in figure 6.2.

A =	a,b,1,5	1 period
B =	c,d,2,4	1 period
C =	a,2	1 period
D =	b,1	1 period
E =	c,3	1 period
F =	d,5	1 period
G =	a,5	1 period
H =	b,c,3,4	1 period
J =	d,1	1 period

Figure 6.2
Requirements for T'

Figure 6.3 gives the corresponding matrix representation of T'.

	1	2	3
A	1	0	0
B	0	0	1
C	0	1	0
D	0	0	1
E	1	0	0
F	0	1	0
G	0	0	1
H	0	1	0
J	0	0	0

Figure 6.3
Matrix Representation of T'

Requirement J is not in the timetable. It cannot be entered directly into period 1 because teacher 1 is already involved in requirement A. Other assignments in T' will have to be moved to allow J to be entered. Figure 6.4 shows a partial tree-interchange which represents one possible sequence of moves that may be attempted. This interchange has two unsatisfied elements - $I_{B,3}$ and $I_{H,2}$. The chains to the negative

elements are shown by the arrows. Figure 6.4(b) is a simplified representation of the interchange which shows only the non-zero elements.

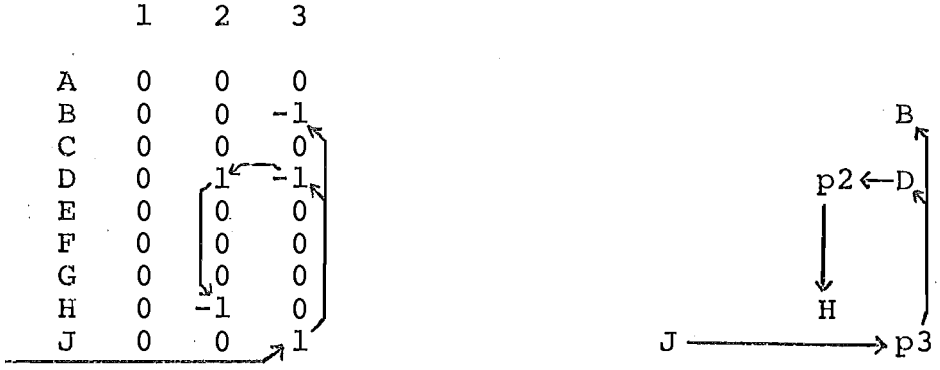


Figure 6.4 Example of Partial Tree-Interchange

When this interchange is added to T', the resulting partial timetable is as shown in figure 6.5.

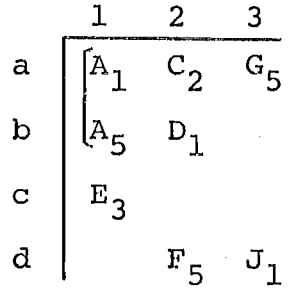


Figure 6.5
T' after Addition of Partial Tree-Interchange

Figure 6.6 is a complete interchange for requirement J.

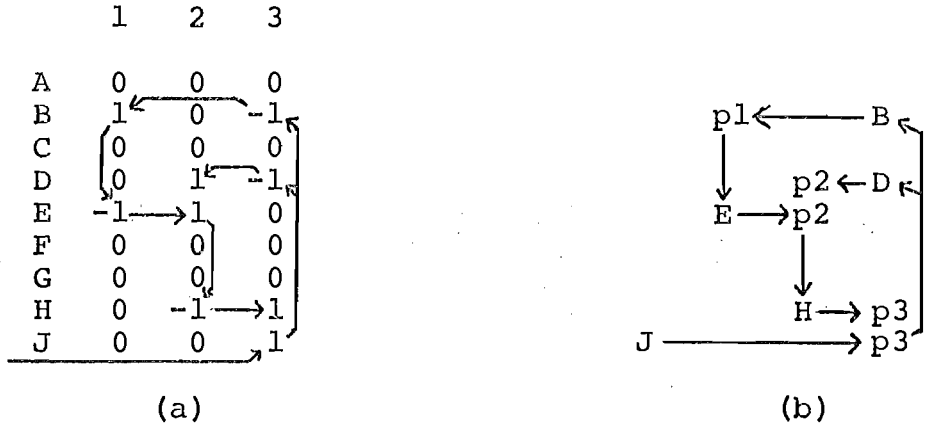


Figure 6.6 Example of Complete Interchange

Addition of this interchange to T' gives a timetable in which all requirements are present (figure 6.7). This interchange is a tree interchange since to every negative element there is

a chain from requirement J, as shown by the arrows.

	1	2	3
a	A_1	C_2	G_5
b	A_5	D_1	H_3
c	B_2	E_3	H_4
d	B_4	F_5	J_1

Figure 6.7
T' after Addition of Complete Interchange

6.5 The Tree Search Stack

The operation of the tree search algorithm requires the use of a stack to record the interchange that has been made to the timetable at any time. Each stack 'word' represents a non-zero element of the interchange matrix and consists of three fields:-

a) A requirement field which may also contain a pointer to a previous stack word;

b) A period field to indicate the 'source' period from which an assignment was taken or the 'destination' period into which it was placed;

c) A 'status' field to indicate the type of stack element represented by the word. This field can have three values:
'U' means that the word represents an unsatisfied element,
'S' means that the word represents a 'satisfied' negative element, i.e. one which has a corresponding positive element in the same row,
'D' means that the word represents a positive 'destination' element.

The structure of the stack is illustrated in figure 6.8. The original requirement is stored in the first word of the stack. The second word is a dummy to ensure proper operation of the algorithm. These two words are initialised before the algorithm is started. Subsequent words store the remainder of the tree-interchange in the following general manner.

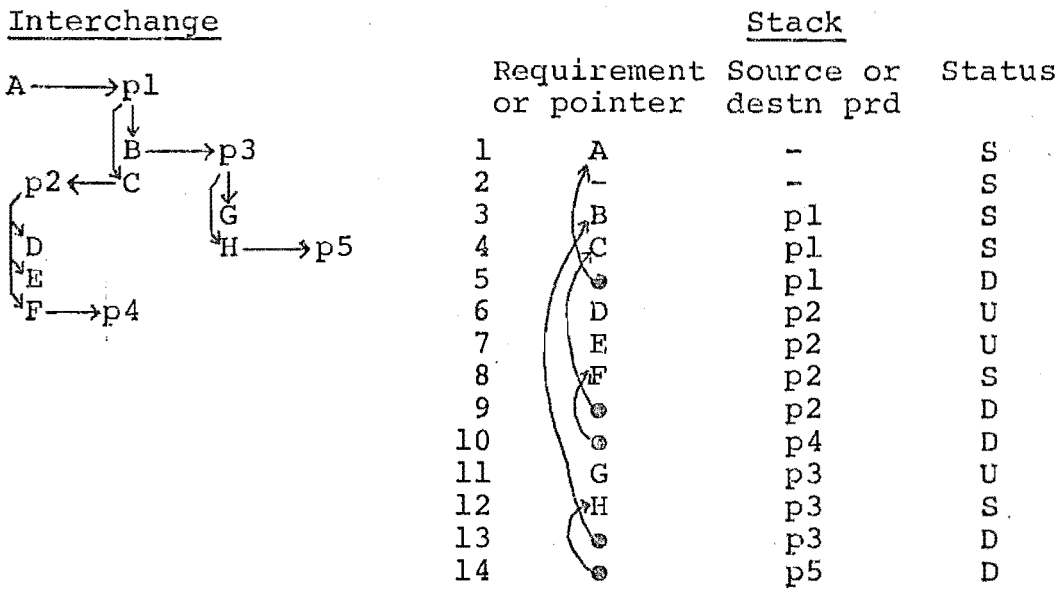


Figure 6.8 Stack Representation of an Interchange

A tree-interchange can be considered to be composed of nodes with associated branches to daughter nodes. The contents of a node and associated branches are shown in figure 6.9(a). The node is represented by a 'node set' of words in the stack (figure 6.9(b)). The last word of the node set is the destination entry representing the positive element in the interchange node. It contains a pointer to its associated source word, as well as the destination period DP_n. The source entries for the displaced assignments are recorded in the remaining words of the node set. The source periods which may in general differ from the destination period are also stored. The status for the source entries is initially set to 'U' for unsatisfied, but when a displaced assignment is reallocated its status becomes satisfied.

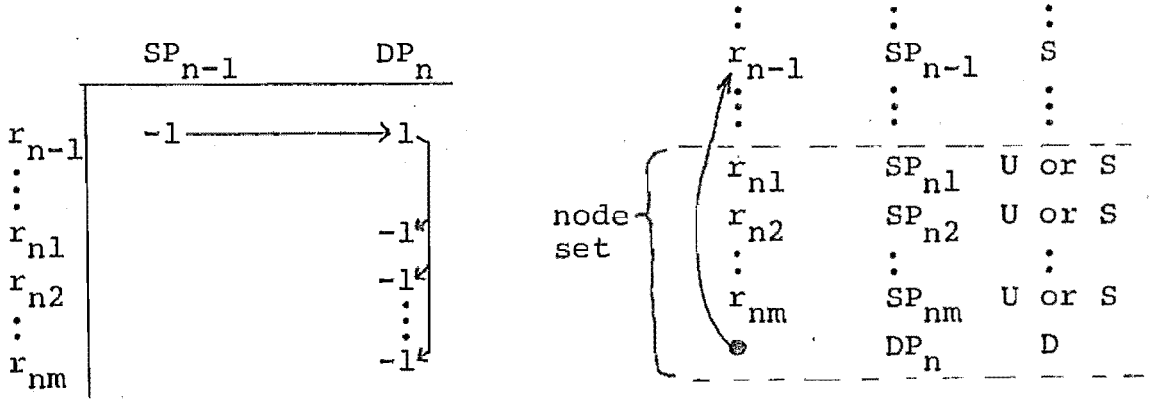


Figure 6.9 Stack Entries for a Node

6.6 The Basic Algorithm

The operation of the algorithm is as follows.

The current requirement is initially the requirement to be put in. A search is carried out to find a period into which the current requirement can be loaded without clashing. If such a period is found, the current requirement is assigned directly. Otherwise, the period search is repeated to find periods with one clashing assignment only, then to find periods with two clashes, then with three, and so on. If a period with the correct number of clashes is found, the current requirement is assigned to that period and the clashes are removed. The assignment is 'fixed' to indicate that it cannot be moved again. ('Fixing' of moved assignments is necessary to prevent endless cycling of assignment moves.) The details of the displacement are recorded on the stack. The requirement associated with one of the displaced clashes is taken as the current requirement and the above search is repeated with it. If no clashes are displaced, the current requirement is loaded directly and a clash which was displaced at an earlier stage and which has not been reassigned is taken as the current requirement. If all displaced clashes have been reentered, then a solution has been found.

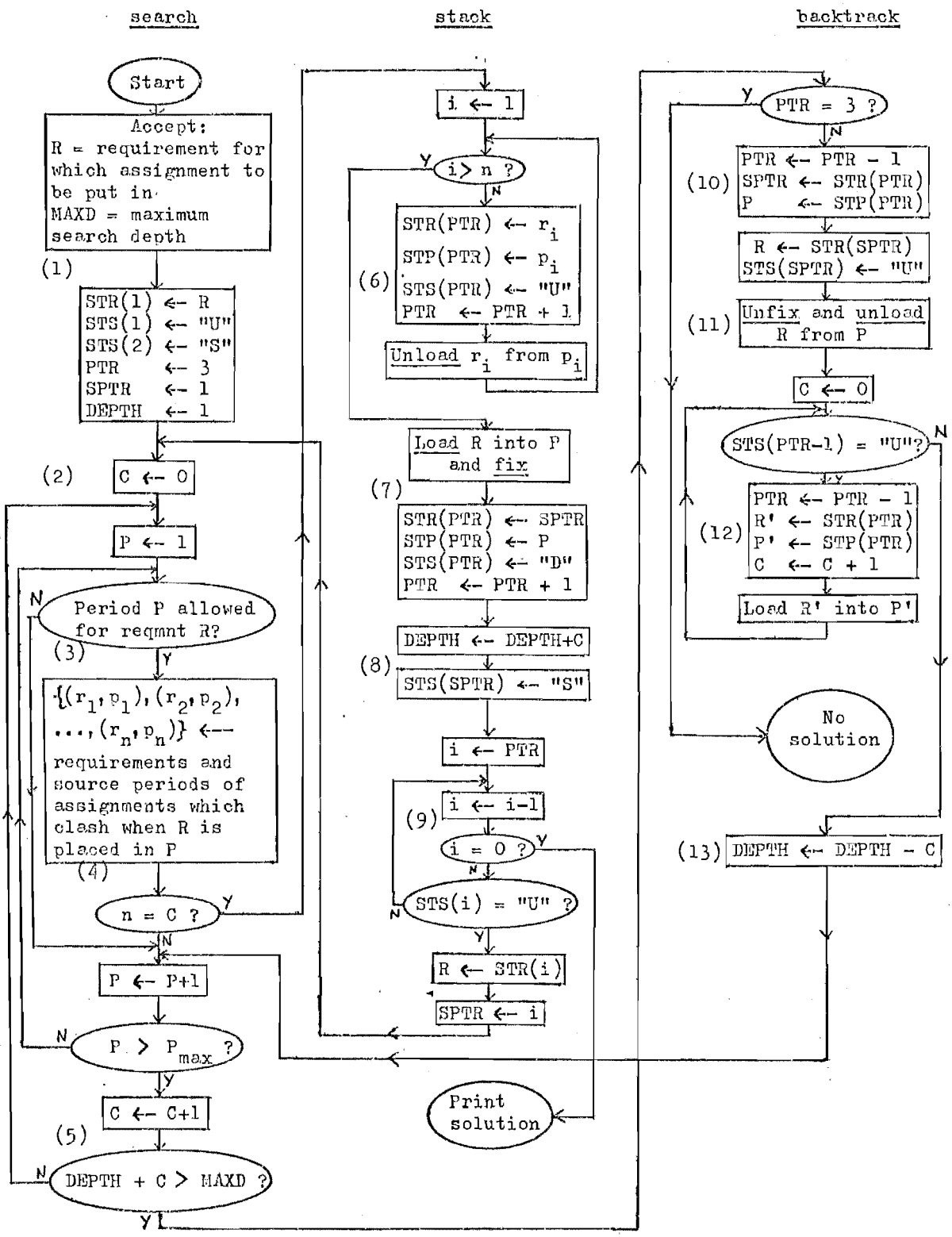
If no solution is found, the stack will continue to grow until a predetermined depth limit is exceeded, at which point backtracking occurs. The last node set on the stack is removed and the displacement operation that it represents is reversed. Searching continues from the period at which this operation occurred. If backtracking reaches the beginning of the stack, the algorithm terminates, having found no solution satisfying the maximum depth criterion.

6.6.1 Details

A flowchart of the algorithm is given in figure 6.10. The algorithm is composed of three parts:-

a) Search - which searches for a suitable period in which to load an assignment.

Figure 6.10 Flowchart of Tree-Search Algorithm



b) Stack - which loads the assignment, displaces the clashes, and records the details on the stack.

c) Backtrack - which reverses the operation of Stack.

The numerical labelling in figure 6.10 is used in the following detailed description.

1) The current requirement R and the maximum search depth MAXD are accepted from the keyboard. The stack is initialised as shown in figure 6.11.

		STR (requirement or pointer)	STP (source or destn prd	STS (status)
SPTR ⟶ (pointer to source word)	1	R		U
	2			S
PTR ⟶ (current stack word pointer)	3			
	4			
	:			

Figure 6.11 Initial State of Stack

2) The variable 'C' is the clash counter. Initially a search is made for periods in which there are no clashes. Then the search is repeated for periods in which there is one clash, two clashes, and so on.

3) Each period P is checked to determine whether or not it is allowed for requirement R. It is not allowed if either there is a restriction imposed by the school stating that R is not allowed in P, or a clashing assignment is 'fixed' and hence cannot be displaced, or the requirement R already has an assignment in P.

4) A list is formed of the clashing assignments, represented by their requirement references and source periods. If the number of clashes equals C, then the period is chosen for stacking. Otherwise the tests (3) and (4) are repeated for the next period.

5) If no periods are found in which the number of clashes is

equal to the current clash number, C is incremented and a test is made to determine whether the maximum depth will be exceeded in the next search. The depth of a tree-interchange is defined as

$$1 - \sum_{r,p} \min(0, I_{rp})$$

which is one more than the total number of -1's (source entries) in the matrix representing the interchange.

6) In the Stack portion of the algorithm, the first task is to remove the clashes. The requirement reference, the source period and the status 'U' are recorded on the stack for each clash unloaded. If there are no clashes, this unloading section is skipped.

7) An assignment is made for the current requirement R into period P and the details are stacked. The assignment is 'fixed' to prevent further moving.

8) The DEPTH is increased by C to indicate that there are now C more -1's in the tree-interchange matrix. The status of the source word for the current requirement is changed to 'S' to indicate that the associated assignment has now been reentered.

9) A new current requirement must now be found. Normally this would be one of the clashes removed in this current stacking operation. However, if there were no clashes then the stack must be searched to find an earlier unsatisfied word. If there are no unsatisfied words, then a solution has been found. Otherwise R is set to the requirement reference of the unsatisfied word and a new search is begun.

10) Backtracking occurs when it is found that the maximum depth will be exceeded on the next period search cycle((3) and (4)). If the stack has only the first two initialisation words, then no solution exists whose depth is within MAXD. Otherwise, details of the last word of the stack are extracted. This word is always a destination word. The requirement reference must, therefore, be accessed via the pointer in this word.

11) R and P have now been set up to the destination entry. The assignment in period P for requirement R is unfixed and removed from that period.

12) The remaining words in the node set are now processed. The end of the node set is indicated by the previous word not being an unsatisfied source word. The requirement and source period for each word are extracted and the displaced assignments are replaced in their source periods. The number of source words is counted to restore the value of C.

13) The DEPTH is reduced by C in accordance with the reduced number of source words on the stack. With P set up to the period into which R was last loaded, searching continues from period P+1.

6.6.2 Example of Operation

	1	2	3	
a	A_1	C_2	G_5	
b	A_5	H_3	D_1	
c	E_3	H_4	B_2	
d		F_5	B_4	$\leftarrow J_1$

Figure 6.12 Example Problem for Application of Tree Search

Consider the problem posed by T' in figure 6.1, reproduced in figure 6.12 for ease of reference. Assignment J, involving class d and teacher 1, is to be entered. The current requirement is therefore set to J and in this example we shall search to a maximum depth of 5.

The initial search with C=0 is unsuccessful. With C=1, period 1 is found to have A as the only clash (because both J and A involve the teacher 1). J is loaded into period 1, displacing A which becomes the current requirement. Period searches for A with C=0 and C=1 are unsuccessful. With C=2, assignments G and D are displaced from period 3. The chosen periods and current requirements are summarised in figure 6.13.

```
                DEPTH after operation

J  1  A          2
   A  3  G,D     4
      D  2  H    5
```

Figure 6.13
First Three Steps of Tree Search

At this point the maximum depth has been reached. A search for H with C=0 is carried out, but then backtracking occurs and D is restored as the current assignment. With DEPTH=4 at this stage, searches with C=0 and C=1 only are permitted. No periods other than 2 are available for D, so backtracking occurs to A.

Execution continues as shown in figure 6.14, with a solution being found after the 19th step.

```
      Indentation =
Step  DEPTH before    DEPTH after operation
      operation
      12345
   1 -J  1  A          2
   2 --A  3  G,D       4
   3 ----D  2  H       5
   4 --A  2  C,H,F     5
   5 -J  2  F          2
   6 --F  1  A         3
   7 ---A  3  G,D      5
   8 -----D  1         5
   9 --F  3  G,B       4
  10 ----B  1  E       5
  11 -----E  3         5
  12 -J  3  D,B        3
  13 ---B  1  E        4
  14 ----E  3          4
  15 ----D  1  A       5
  16 ----D  2  H       5
  17 ----E  2  H       5
  18 -----H  3         5
  19 -----D  2         5  -  solution:  J  3  D,B
                                        B  1  E
                                        E  2  H
                                        H  3
                                        D  2
```

Solution timetable:

	1	2	3
a	A_1	C_2	G_5
b	A_5	D_1	H_3
c	B_2	E_3	H_4
d	B_4	F_5	J_1

Figure 6.14 Execution of Tree Search

6.7 Handling the Complexities of the Real Problem

With no change in the basic structure, the tree search algorithm can handle most of the complexities of the practical timetabling problem, provided that these can be expressed as 'yes-no' conditions. This is important since the conditions form the basis on which decisions must be made, such as whether or not to test a period, or to displace a clash. Ill-defined constraints, which require numerical weights for proper specification, cannot be handled unless either thresholds are used or the constraints are approximated by simpler yes-no functions. The latter technique is utilised in the current implementation.

6.7.1 Double and Triple Periods

A multiple-period lesson is considered as a single assignment which occupies n periods of the timetable. Double and triple period assignments can be associated with requirements in a similar manner to single-period assignments. As well as $N_r^{(s)}$ giving the number of single-period assignments associated with requirement r, $N_r^{(d)}$, $N_r^{(t)}$, ... can be included in the initial data to give the numbers of double-period, triple-period, etc assignments respectively. When an n-period assignment for r is loaded into timetable period P, T_{rp} is set to 1 for p = P, P+1,P+2,...,P+n-1, and the assignment will displace clashes from all these periods. (The total number of periods in the completed timetable for a requirement is given by

$$\sum_p T_{rp} = N_r^{(s)} + 2N_r^{(d)} + 3N_r^{(t)} + \ldots)$$

These displacements can be recorded on the stack in exactly the same way as before, except that, unlike the example in figure 6.8, the source periods of the displaced clashes will not necessarily be the same as the destination period for the multiple. Multiple periods are subject to a special restriction imposed by the school — they must not cross breaks between certain defined pairs of adjacent periods. Violation of this restriction is prevented simply by disallowing the n-1 periods before each break for each n-period assignment.

6.7.2 Classrooms

Resources in short supply, such as specialist rooms, must be considered during timetabling. If there is only one room of a particular type in the school, that room can be handled in exactly the same way as a class or teacher. Two assignments requiring that room will clash and cannot be assigned to the same period.

A different problem arises when there is more than one room of a particular type. If, for example, there are two Technical Drawing rooms in the school and three assignments which otherwise do not clash each require one T.D. room, then any two assignments can appear together in a period, but not all three. In general, if there are L_k rooms of a particular type k available and the rth requirement requires X_{rk} of them, then requirements $r_1, r_2, r_3, \ldots, r_m$, which do not otherwise clash with one another, can all have assignments in one period only if

$$\sum_{j=1}^{m} X_{r_j k} \leq L_k .$$

Consider the problem of assigning requirement r_o requiring $X_{r_o k}$ rooms of type k into a period P which already contains assignments from requirements $r_1, r_2, \ldots, r_m$. If

$$X_{r_o k} + \sum_{j=1}^{m} X_{r_j k} > L_k ,$$

then a subset of $\{r_1, r_2, \ldots, r_m\}$ will have to be removed from period P in order to restore the room requirements to less than L_k. In general, there will be a choice of such subsets. Let the set of all such subsets be Q. When r_o is entered into the period, we wish to keep the number of displaced assignments to a minimum to avoid unnecessary tree searching. Hence we are interested in finding the set of minimal subsets $Q' \subset Q$ in which, for all $q_1, q_2 \in Q'$, $q_1 \not\subset q_2$. A special subroutine can be added to the main tree-search routine to find minimal subsets and present them as lists of clashes. However, this requires complex and cumbersome modifications to the backtracking routine and to the organisation of the stack to ensure that all minimal subsets in Q' are tested. It was

therefore decided to employ a simpler strategy which requires testing of more than the minimal sets of Q', but which does not require any modification to the main algorithm and which does not entail the testing of every single set in Q. Two such strategies exist. Both rely on the operation of the tree-search algorithm itself for their proper functioning.

The first strategy is to displace all assignments requiring one or more rooms of the type that is causing the clash. The tree-search algorithm later replaces some of these assignments back into the period from which they were displaced.

The complete set $\{r_1, r_2, \ldots, r_m\}$, where $X_{r_j k} > 0$ for $j = 1,2,..,m$ is removed from period P and recorded on the stack before r_o is put in. The requirement of one of these displaced assignments, say r_1, is selected by the tree search as the current requirement and is tested in all periods, including P. If $X_{r_o k} + X_{r_1 k} \leqslant L_k$, then P is free for r_1 and direct loading can take place. Another displaced requirement, say r_2, now becomes the current requirement and is also loaded into P if $X_{r_o k} + X_{r_1 k} + X_{r_2 k} \leqslant L_k$. Displaced assignments are replaced in this source period until the sum of the room requirements for some k exceeds L_k when r_s is tested in P. The period is now no longer free for r_s and the tree search attempts to find another period for r_s. If the search for r_s or for any other of the remaining displaced assignments $r_{s+1}, \ldots, r_m$ is unsuccessful, backtracking occurs and r_{s-1} is removed from P. If r_{s-1} is successfully placed elsewhere, r_s will again be tested in P as well as in other periods and will be entered if

$$X_{r_o k} + \sum_{j=1}^{k-2} X_{r_j k} + X_{r_s k} \leqslant L_k.$$

If no solution for r_{s-1} is found, then r_{s-2} is removed and an attempt is made to place it elsewhere. The process continues until either a solution is found with some of r_1 to r_n replaced in P and the remainder positioned elsewhere, or no solution is found and r_o is removed.

The second strategy is to remove only enough of the first few assignments in the period P to eliminate excessive classroom requirements. On later searching, the tree search will, at some stage, re-enter a displaced clash into P and displace further assignments.

When r_o is entered into P, which contains $\{r_1, r_2, \ldots, r_m\}$, the requirements $r_1, r_2, \ldots, r_s$ are removed, where s is the smallest value such that

$$X_{r_o k} + \sum_{j=s+1}^{m} X_{r_j k} \leq L_k .$$

The requirement of one of the displaced assignments, say r_1, becomes the current requirement. This is tested in all periods including P, in which it may displace a further set of assignments in a similar manner (only a sufficient number of the first few non-fixed assignments to eliminate excessive classroom requirements). If all these assignments can be reallocated properly, then r_2 becomes the current requirement and may be either replaced in P or moved to another period. The process continues until either all of r_1 to r_s and their subsequent displacements have been repositioned, or no solution is found.

Fixed classroom clashes cannot, of course, be displaced. If the subset $\{r_{i_1}, r_{i_2}, \ldots, r_{i_q}\}$ is fixed in P and

$$X_{r_o k} + \sum_{j=1}^{q} X_{r_{i_j} k} > L_k \quad \text{for some } k,$$

then P must be disallowed since classroom restrictions cannot be satisfied even when all unfixed clashes have been removed. If the sum is less than or equal to L_k for all k, then either of the above strategies may be used on the non-fixed assignments in the period, with L_k replaced by

$$L_k - \sum_{j=1}^{q} X_{r_{i_j} k} .$$

The first strategy for classroom handling has been tested and has been found adequate for practical timetabling problems in which the total number of classrooms of any type is small. However, if the number of rooms is large (for example, if the total number of classrooms in the school must be considered), then the simple methods described above are likely to be inefficient and it may become necessary to include heuristics.

6.7.3 Distribution

An important 'human' requirement for an acceptable timetable is that lessons of a particular subject for a particular class must be well spread through the week. For example, if there are 5 periods of a subject, then the best distribution is one period of that subject per day. Two periods on one day and none on another is less satisfactory, but still generally acceptable, while three periods on one day is generally unacceptable. The acceptability of distribution patterns varies with the subject and the number of periods required. In addition to the number of periods of the subject per day, the relative positioning of periods within a day is important. For some subjects, if there are two periods on one day, then they must be adjacent so as to form a double. For others, they must be well separated — preferably one in the morning and the other in the afternoon.

As mentioned before, constraints having varying degrees of acceptability cannot easily be handled by the basic tree-search algorithm. It is necessary to simplify distribution requirements into conditions which determine whether a pattern is simply either 'acceptable' or 'unacceptable'. Such simplified constraints may not allow accurate description of the school's true preferences for certain patterns over others, but in an interactive environment this is not a disadvantage since the timetabler has the option of changing distribution constraints at any time during construction. Yes-no type constraints are also considerably easier to manage than weighted constraints.

The distribution-checking program forms part of the clash-list-generating and period-checking components labelled by (3) and (4) in figure 6.10. It checks the distribution that results

when R is placed in P and returns one of the following verdicts:

a) R is not allowed in any period of the current DAY.

b) R is not allowed in period P.

c) R can be put into P without making the distribution unacceptable.

d) R can be put into P, but 'distribution clashes' must be removed.

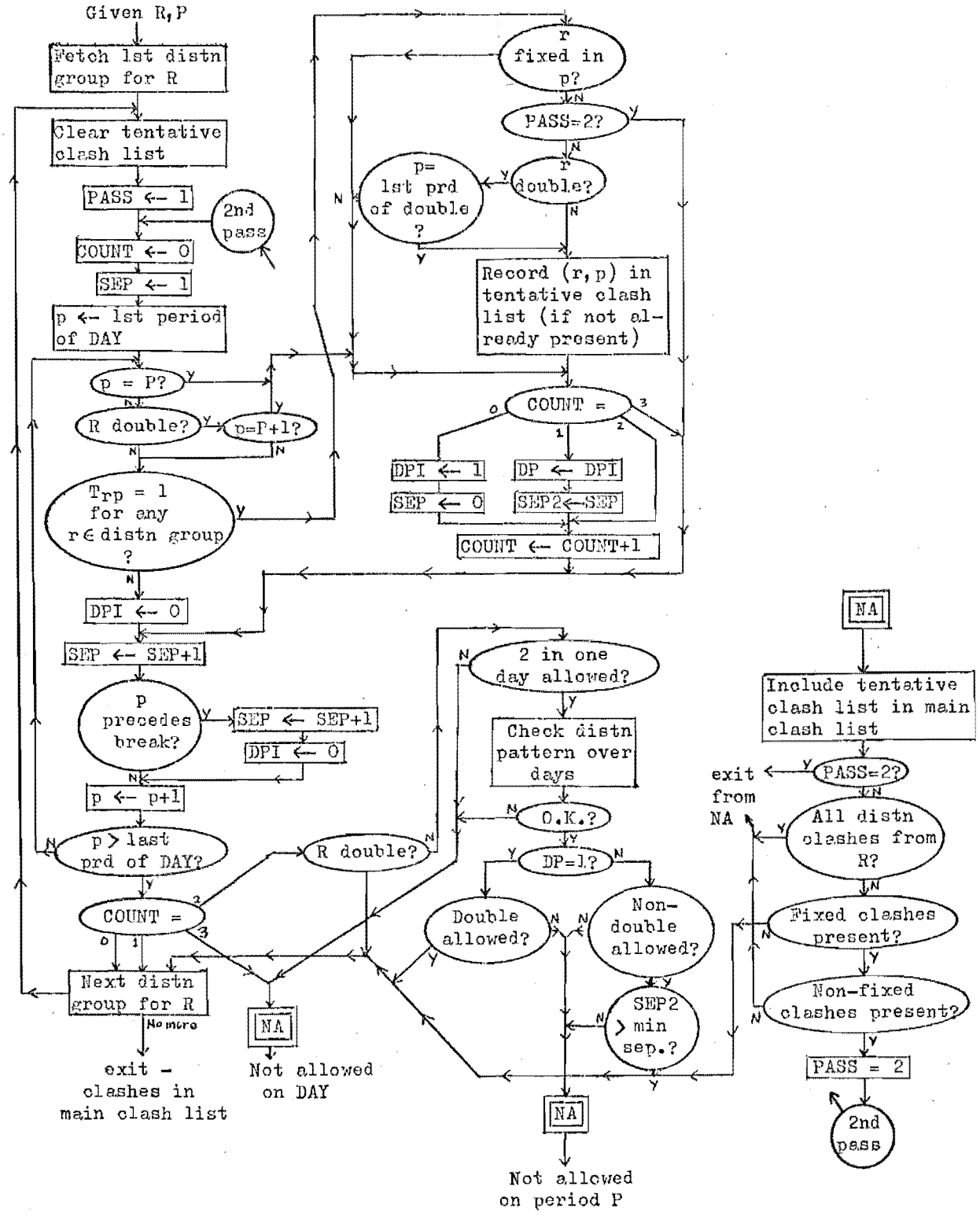
Distribution clashes are assignments already in the time-table which, if allowed to remain when R was entered, would result in unacceptable distribution patterns.

The lessons in a particular subject for a particular class are not in general all included in one requirement but may be distributed among several requirements. All members of such a 'distribution group' must be considered when examining the distribution of the subject. Distribution constraints apply to distribution groups and not to individual requirements. Furthermore, since R may in general be a block consisting of several subjects for several classes, it may be involved in more than one distribution group.

The distribution-checking routine (figure 6.15) carries out the following procedure for every distribution group in which R is a member. If the procedure finds that the distribution is acceptable for one group (after removing distribution clashes), then it is repeated for the next group. Otherwise an immediate exit is made.

a) Initially, a scan is made of the periods of the DAY containing P to determine:
   (1) the number of periods in DAY in which members of the distribution group are present (this number includes the contribution to be made by R).
   (2) the relative positioning of these periods, if there is more than one, i.e. whether or not they form a double

Figure 6.15 Distribution-Checking Routine



and the number of periods and breaks separating them. Assignments found in DAY which are members of the distribution group and which are not fixed are recorded in a tentative clash list. No assignment is recorded more than once.

b) The acceptability of the distribution depends initially on the number of periods found in (a)(1). In the current implementation, the acceptability rules are as follows:

If there is one period in DAY, then the distribution is always acceptable.

If there are three or more periods in DAY, then the distribution is always unacceptable for DAY, no matter what period is chosen.

If there are two periods in DAY, then the acceptability depends on

(1) the number of periods on other days. One of the parameters of the constraint data associated with the distribution group is the maximum number of days with two periods permitted. This number depends on the total number of assignments in the distribution group as well as on the subject concerned. For example, if the total is 6, then a maximum of one day with two periods represents a very rigid constraint since it allows only the optimal distribution: 2-1-1-1-1. A maximum of two days allows also the less preferred distribution of 2-2-1-1-0. If the distribution of the periods of the distribution group (including the contribution to be made by R) is found to be unacceptable for any reason, then it is unacceptable for DAY.

(2) the relative positioning of the periods in DAY. Other parameters in the constraint data indicate whether or not doubles are allowed for members of the distribution group and the minimum gap between the periods. If the actual positioning is not consistent with these parameters, then the distribution is unacceptable for the period P.

c) If the distribution is found to be acceptable in (b), then the tentative clash list is deleted. Otherwise the elements of this list are added to the main clash list.

d) If the distribution is found to be unacceptable, then the following action takes place:
(1) If all the assignments found in DAY are from the requirement R, then the routine exits with a verdict of 'not allowed on DAY' or 'not allowed in P' depending on the reason why the distribution was unacceptable.
(2) If all the assignments found in DAY are fixed, then the routine exits with the same verdict as in (1).
(3) If none of the assignments found in DAY are fixed and not all of them are from the requirement R, then the routine continues to the next distribution group, after recording the assignments on the clash list.
(4) If some of the assignments found in DAY are fixed, then the complete distribution checking algorithm is repeated, with the non-fixed assignments considered as absent. No new entries to the clash list are made. The purpose of doing this is to ensure that the distribution of the new assignment together with the fixed assignments only is acceptable. If so, the routine continues to the next group as in (3), otherwise it exits as in (2).

Ideally, the distribution routine should record as clashes all assignments in the timetable whose requirements are in the distribution group, if the distribution is found unacceptable. This is because distribution acceptability is a function of the positions of all the assignments involved. For example, whether or not two periods of a subject are allowed in DAY depends on the number of other days with two periods of the subject. In practice, however, this 'ideal' procedure would impose an extremely heavy burden on the tree search, as in the case of the simple classroom procedures when the number of rooms of one type is large. The simple heuristic of recording only the assignments within DAY as clashes is therefore necessary. A second heuristic — that of not displacing the assignments in DAY if they are all from requirement R — was included for the following reason.

Suppose that a solution is found in which an assignment from R is placed in period P_1, causing an assignment from R to be removed from P_2 as a distribution clash. This assignment is replaced in P_3 (figure 6.16(a)). Then a similar solution can usually be found simply by placing R directly in P_3 (figure 6.16(b)).

Figure 6.16

In the first solution, an assignment is being moved from P_2 to P_3 unnecessarily (assuming that no other assignment moved in the interchange clashes with R in P_2). Thus the speed of the tree search can be improved with little risk of missing solutions by introducing the second heuristic.

As important as subject distribution is the distribution of teaching and of non-teaching periods for each teacher. However, no attempt has been made to mechanise teacher distribution checking in the program, since

a) the number of teachers in the school is far less than the number of class-subjects. It is easier for the timetabler to control the distributions of 50 to 60 teachers interactively than to control the distributions of 200 or so class-subjects.

b) the positioning of free periods for a teacher becomes important only when the teacher's teaching periods have nearly all been allocated. Thus it is not necessary to maintain a constant check on the distribution for every teacher.

c) teacher distribution restrictions as specified by the school are generally of an ill-defined and variable nature. It is not possible to get adequate representations

of these in a simple checking algorithm.

6.8 Application to Timetabling

The tree-search algorithm as described together with facilities for handling classrooms and distribution formed an important part of the interactive timetabling program which was used for the construction of three timetables for Riccarton High School. In this program, the tree-search routine accepted the requirement reference and the maximum depth from the keyboard. If it found a solution, the contents of the stack were printed out. If no solution could be found within the current depth, the depth was increased by three and searching was repeated. During the timetable constructions, the search was nearly always started with depth initialised to 4; this was increased automatically to 7, then to 10, and so on. The times to find solutions varied considerably, but as an indication of typical times for complete searches at various depths (using the EAI 640 computer), we give:

Depth = 4	10 to 20 seconds
Depth = 7	30 seconds to 3 minutes
Depth = 10	more than 10 minutes

As in other combinatorial search programs, the search time increases exponentially with the size of the problem. What is required are techniques to prune the search tree to prevent repetitive searching and to eleminate branches that do not lead to solutions. Particular cases of such unnecessary searching are the following:-

a) The operation of the algorithm is such that it completes a branch of the tree-interchange before starting another. If it fails to complete a branch (because of the depth limit), then the algorithm backtracks to an earlier branch that has been completed. The last assignment in that branch is removed and searching continues on that branch in an attempt to find a new series of moves which complete the branch. In most cases it is unnecessary to continue a search on a completed branch in this manner. It is necessary only when the assignments moved and fixed in

the earlier branch prevent the later branch from being completed.

b) When applied to the problem of entering the last assignment of a class, the algorithm does not have the human facility of being able to 'see' that it is necessary to fill the empty period with one of the class's assignments before a solution can be found. Instead, it becomes involved in a massive search of permutations or rearrangements of the class's assignments already in the timetable.

In some cases, manual assistance had to be given since the algorithm would have taken an impractically long time to find a solution. This manual assistance consisted of making a few moves by hand before using the algorithm to replace the resulting displaced assignments.

6.9 A Modified Tree-Search Algorithm

Modifications can be made to the basic algorithm to eliminate unnecessary searching of type (a) in the last section. The first and most important modification is to the backtrack portion of the algorithm.

Suppose that, immediately before backtracking, the tree-interchange represented by the stack is as shown in figure 6.17, with C as the current requirement.

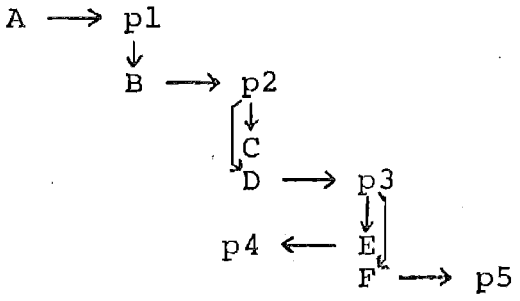


Figure 6.17

In the original algorithm, backtracking would remove F from p5 and searching would continue with F as the current requirement. In the modified version, the backtracking step is repeated until the source word for the current requirement

is removed from the stack, i.e. F, E, D and C are restored, B is removed from p2, and searching continues with B as the current requirement. Thus we acknowledge that, in most cases, if no complete branch could be found for C then there is no point in attempting to alter the branch that has already been completed for D.

The exception to this rule arises when, at some stage during the search for a branch for C, a clash with a fixed assignment in the branch from D occurs (figure 6.18).

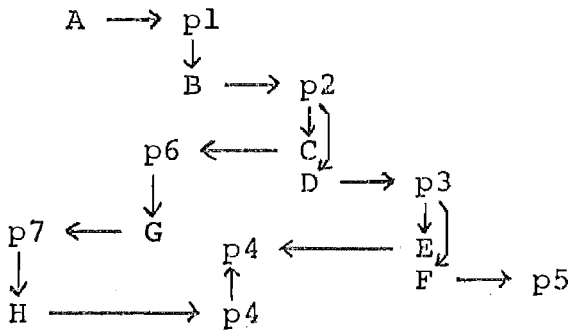


Figure 6.18

In this case, if H is entered into p4, then it clashes with E which has already been entered and fixed in p4. Now it may be that if E were not present in p4, then H could be entered into p4 directly. If E could be located elsewhere, then a solution has been found for A. Thus the presence of the branch from D is actually preventing a solution.

To overcome this problem, a second modification is made to the basic algorithm: to unfix all assignments on completed branches and leave fixed only those assignments on the chain between the original requirement and the current requirement (the main chain). Thus, in the example of figure 6.18, the fixed assignments would be those labelled with an asterisk (*) in figure 6.19.

When H is now put into p4 it can displace E from that period. E can be moved to an alternative period, giving a solution which would not have been found if the second modification had not been made (figure 6.20).

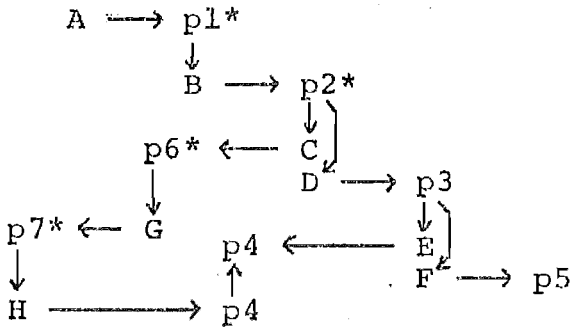


Figure 6.19

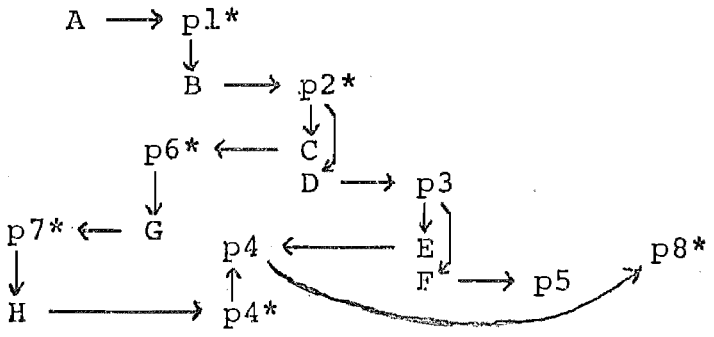


Figure 6.20

Looping is prevented by the fixing on the main chain. No danger of cycling of interchanges is introduced by unfixing the assignments on completed branches.

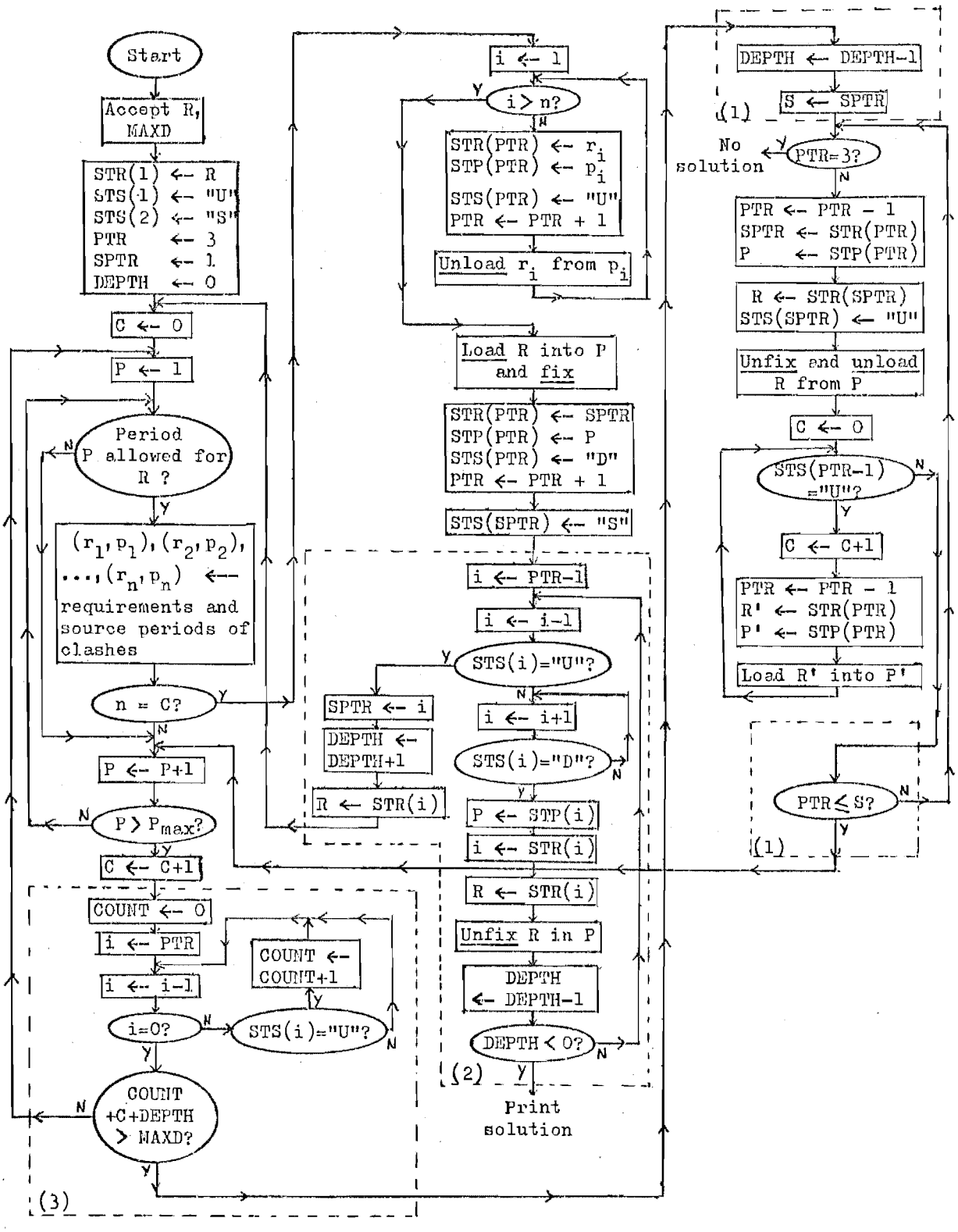
The third modification is to redefine DEPTH as the length of the main chain in the interchange, in terms of the number of destination entries. The DEPTH of the example above is 6 since the destination periods p1, p2, p6, p7 and p4 are on the main chain.

6.9.1 Details of the Modified Tree-Search Algorithm

The modified algorithm is given in flowchart form in figure 6.21. Portions which differ from the basic algorithm (figure 6.10) are enclosed in dotted lines and are labelled:-

1) Backtracking is repeated until the source word for the current requirement (pointed to by S) has been deleted. In this way completed branches which start from the same destination entry as that which displaced R, are deleted. However, the DEPTH is only decreased by one since the effect of the multiple backtrack operation is to make one

Figure 6.21 Modified Tree-Search Algorithm



step backwards in the main chain.

2) After a branch has been completed (indicated by the absence of unsatisfied source words in the last node set on the stack) this section retraces the branch, unfixing destination entries on the way until it reaches an unsatisfied word. The requirement of this unsatisfied word is taken as the new current requirement.

3) With the DEPTH redefined in terms of destination entries instead of source entries, the clash number C should, in principle, not be involved in the test for excessive depth. However, in practice it was found necessary to reintroduce C into the excessive depth test to 'taper' the search, since otherwise a great deal of time was spent in testing periods with large numbers of clashes when the stack was near the depth limit. The tapered limit reduces the maximum number of clashes permitted as the depth limit is approached and thereby reduces search time.

Another change resulting from the redefinition of DEPTH is the introduction of explicit counting of unsatisfied source entries in the stack. The count, which was previously implicit in DEPTH, is used in the depth limit test to further taper the search according to the number of entries in the stack that remain to be satisfied.

6.9.2 Application of the Modified Algorithm to Timetabling

The modified algorithm was used in place of the basic algorithm in the program used for the February 1974 timetable construction for Riccarton High School. The average search times at each specified maximum depth were slightly greater than before, but since the maximum depth no longer limited the total number of assignments moved (it affected only the length of the main chain) large interchanges comprising many relatively short branches could be found within a reasonable time. This was found to be of considerable advantage in the block-entering stage of timetabling, since block interchanges are characterised by having many short branches. The largest block interchange found by the algorithm was the one shown in figure 6.22.

1	#I	1/3	%C
2	%C	1/2	&M
3	&M	4/6	#C,#M,+F,#I,&M
4	&M	1/1	
5	#I	4/1	
6	+F	2/1	&M,+T,+T
7	+T	4/6	
8	+T	4/7	
9	&M	5/3	#I
10	#I	2/1	
11	#M	4/5	+I
12	+I	4/3	+A
13	+A	4/1	+X
14	+X	4/5	@I
15	@I	4/6	@U
16	@U	4/1	
17	#C	3/5	@R
18	@R	2/1	#I,#C,@C
19	@C	2/7	@F
20	@F	3/5	
21	#C	2/6	&E
22	&E	2/7	#R
23	#R	2/4	#F
24	#F	2/1	
25	#I	2/7	&E,+I
26	+I	2/3	+A
27	+A	2/7	
28	&E	2/6	#C
29	#C	4/2	@R
30	@R	3/4	

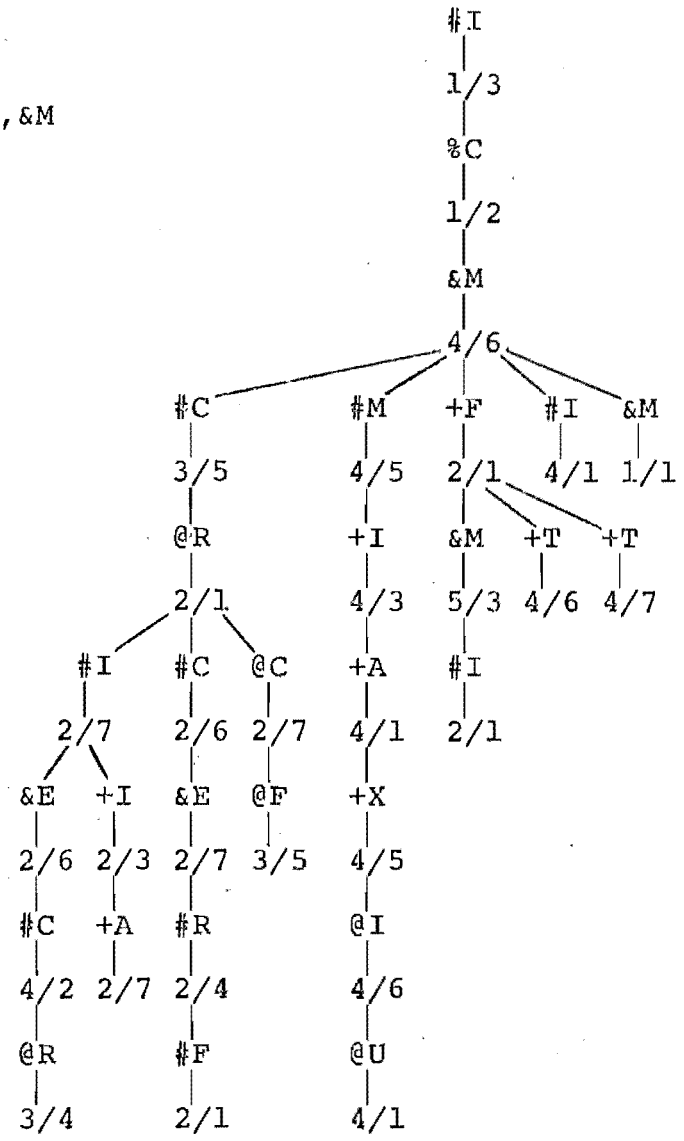


Figure 6.22 Largest Block Interchange

Requirements are represented by two-character codes in which special characters are permitted, and periods are represented in the form (day number)/(period number). The maximum depth at the time was 10. This solution was found within 6 minutes of the start of the search at depth 10.

As was demonstrated in section 6.8, unlike the basic algorithm, the modified algorithm is able to move an assignment more than once within one tree-interchange. In this particular example, this has happened three times:

#I from 5/3 to 2/1 then to 2/7
#C from 2/1 to 2/6 then to 2/7
&E from 2/6 to 2/7 then to 2/6.

The net number of assignments moved is therefore 26.

Less improvement was experienced when the new algorithm was applied to the entering of singles. Interchanges found by the algorithm rarely exceeded 10 assignments in size, especially in the later stages of timetable construction. Manual assistance was still sometimes necessary when the algorithm was used to enter the last assignment of a class. This may be because singles interchanges are characterised by less branching but longer chains than interchanges of blocks. The depth limit therfore has a much greater effect on the overall size.

The need for techniques for improving the performance of the tree search for singles is still apparent. The following sections describe two search techniques which have been tested with this aim in mind.

6.10 Explicit Storage of the Search Tree

The tree searched by the basic algorithm is defined as follows:

a) The root node represents the timetable in its original state with the initially specified requirement as the current requirement.

b) The branches from a node represent the possible allowed periods into which the current requirement may be entered.

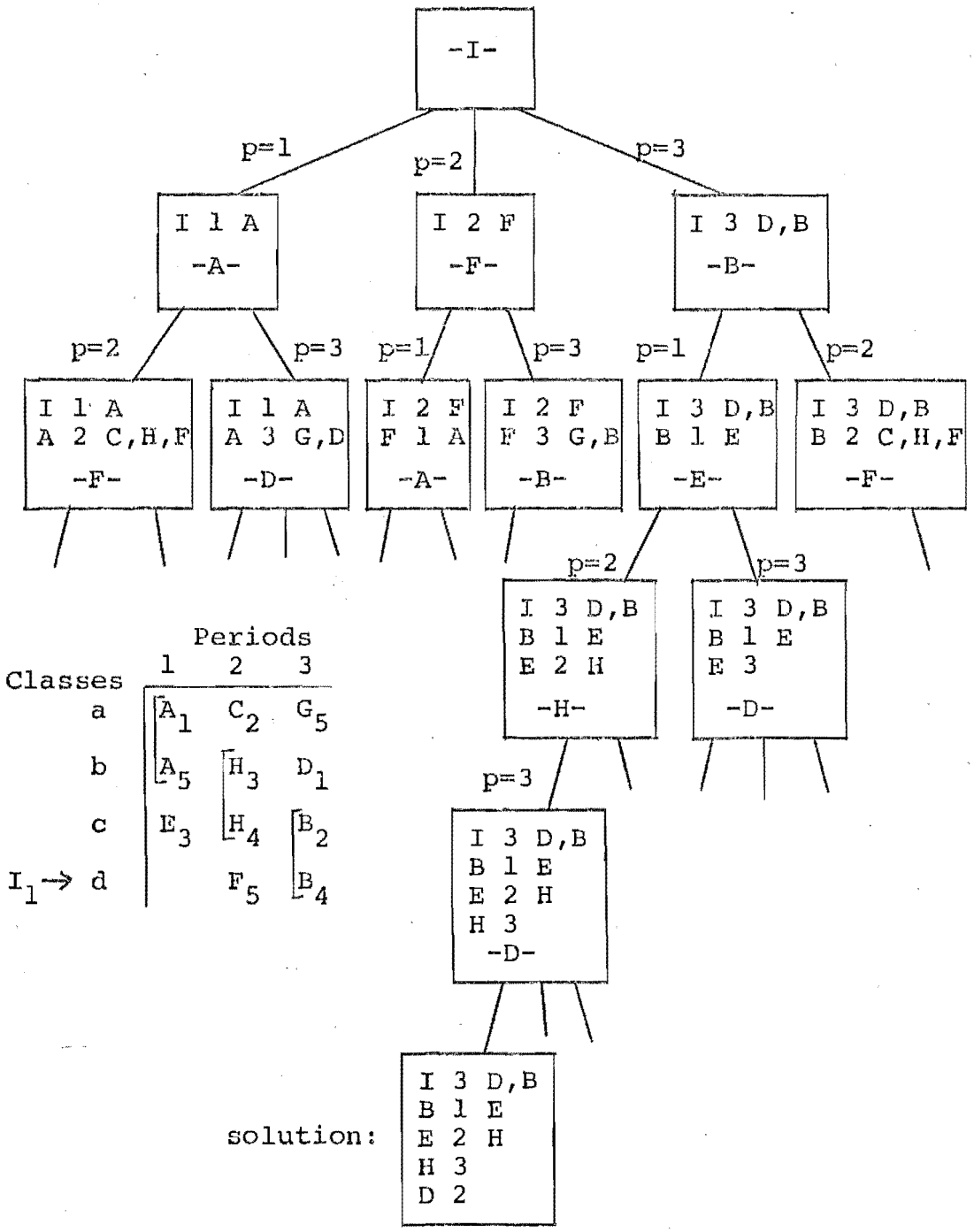


Figure 6.23 Search Tree of Interchanges

c) The node at the end of a branch represents the state of the timetable which results when the current requirement is placed in the period represented by the branch and clashes are displaced. The new current requirement is chosen from amongst the displaced clashes (or previous unsatisfied source entries if there are no clashes) in the manner of the stack portion of the basic tree-search algorithm (figure 6.10).

Figure 6.23 shows the search tree for the example in figure 6.12. In the algorithms to be described this tree is stored explicitly. To save space the only information that is actually stored for each node is

a) the period associated with the branch connecting the node to its predecessor, and

b) a pointer to the predecessor node.

Information on the interchange and the current requirement associated with a node is obtained via a subroutine SETT(NODE) which, by using the period information in the tree, makes the necessary changes to the timetable to set it to the state represented by NODE. If the timetable is already in the state represented by another node, say NODE', then the subroutine backtracks the timetable to the common predecessor of NODE and NODE', and advances it down the tree branch to NODE (figure 6.24).

This operation is controlled by a stack identical to that used in the basic algorithm (section 6.6). After the operation, this stack gives details of the interchange and the current requirement is also available.

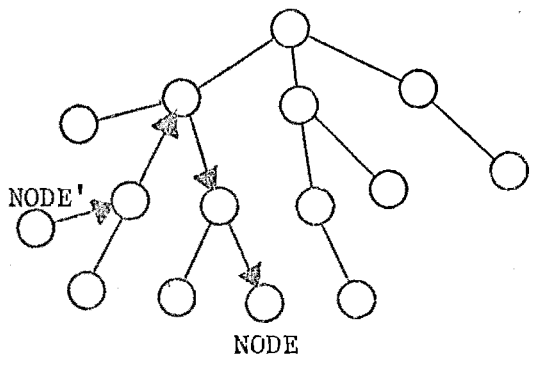


Figure 6.24

Operation of SETT(NODE)

6.11 Best Node Expansion

The method of best node expansion requires a 'value' to be stored with each node. The value estimates the 'difficulty' of re-entering the unsatisfied requirements back into the timetable. Starting with the root node, the method 'grows' the search tree by iteratively choosing the node with the lowest 'value' and generating its descendants. By always choosing the 'easiest' node, the method should on average find a solution after examining fewer nodes than would a systematic exhaustive search. The technique is similar to that used for searching goal trees in theorem proving (Slagle, 1971) and is equivalent to the 'branching' portion of branch-and-bound (Lawler and Wood, 1966).

6.11.1 Algorithm

The tree is represented by two arrays:

PRD(NODE) = period associated with the branch leading to NODE
PTR(NODE) = pointer to predecessor node

The function VALUE(NODE) calculates the difficulty value of the interchange associated with NODE. The set UNEXP contains all nodes which have so far not been expanded (i.e. have not had successor nodes generated). NMAX is the numeric reference of the most recently added node.

1. Initialise: R ← requirement to be put in,
   NMAX ← 0, NODE ← 0.

2. P ← 1.

3. Test requirement R in period P. If it can be entered directly and there are no other unsatisfied requirements in the interchange represented by NODE, then print out solution and exit.

4. If P is forbidden for R, go to 6.

5. Create new node on tree: NMAX ← NMAX + 1
   UNEXP ← UNEXP∪{NMAX}
   PRD(NMAX) ← P
   PTR(NMAX) ← NODE

6. $P \leftarrow P + 1$; if $P \leqslant P_{max}$ go to 3.

7. NODE ← node ε UNEXP for which VALUE(node) is minimum.

8. Set timetable to state of NODE: SETT(NODE).
   R is set to new current requirement.

9. Delete NODE from set of unexpanded nodes:
   UNEXP ← UNEXP - {NODE}

10. Go to 2.

6.11.2 The VALUE Function

The choice of this function is important, since it must reflect accurately the true 'difficulty' of completing the interchange represented by the node and it must be quick to compute. A simple function is

VALUE(node) = number of unsatisfied entries in the interchange represented by 'node'.

This value function does not distinguish between an interchange in which there are three singles unsatisfied and one in which there are three blocks unsatisfied. Yet, obviously the latter is more difficult since displaced blocks are more difficult to re-enter. Thus it is necessary to take into account the 'size' of the requirement displaced, as in the following definition:-

$$\text{VALUE(node)} = \sum_{\substack{\text{unsatisfied} \\ \text{entries in} \\ \text{interchange}}} \text{number of items involved in requirement for unsatisfied entry}$$

This was the function chosen for the test of the algorithm. Examples of other possible value functions are given in Platts (1973) section P.3.4.3.

The best-node-expansion algorithm has the advantage over the depth-first algorithm that it is possible to handle weighted distribution constraints. This is achieved simply by adding to VALUE(node) a measure indicating the overall adverse effect that the interchange represented by 'node' has on the dist-

ribution. If the interchange improves the distribution, then this contribution is negative. By doing this, nodes representing interchanges which preserve distribution quality are chosen in preference to those that do not. The importance of distribution quality can be adjusted by varying the relative weights of 'difficulty' and 'adverse effect on distribution' in the total VALUE.

6.11.3 Application of Best Node Expansion to Timetabling

The algorithm was tested using the second VALUE function above with the distribution contribution included. With singles, it was found to have the fault of both of the depth-first tree-search algorithms — becoming involved in a search of permutations of requirements for a class. The VALUE function did not encourage the placing of one of the class's assignments in the empty period for the class. With blocks, the VALUE function was misleading since the difficulty of a block depends on other factors as well as the number of items involved. An improved VALUE function is required if unnecessary searching is to be eliminated.

6.12 Superset Elimination

Much of the apparently unnecessary searching carried out by the foregoing algorithms can be attributed to the following cause.

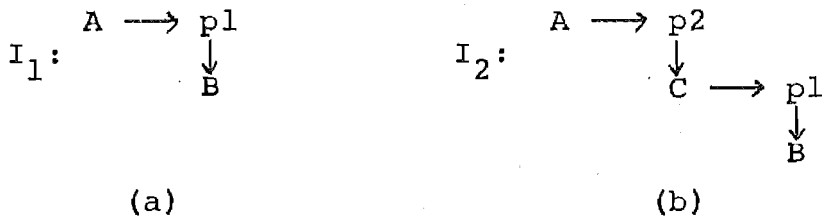


Figure 6.25 Interchanges displacing B

Consider the example in figure 6.25. After having found the interchange of 6.25(a), the algorithm carries out a search for B. If the algorithm then finds another interchange that also displaces B from p1, say 6.25(b), then it repeats the search for B. The procedure followed in this repeated search is almost identical to that followed in the first search since there is very little difference between the timetables in the

two cases. Thus, if a solution can be found by searching from I_2, then there is a good chance of a similar solution being found from I_1 and the extra searching from I_2 is therefore unnecessary. For example, if figure 6.26(a) is a solution, then 6.26(b) is also a solution.

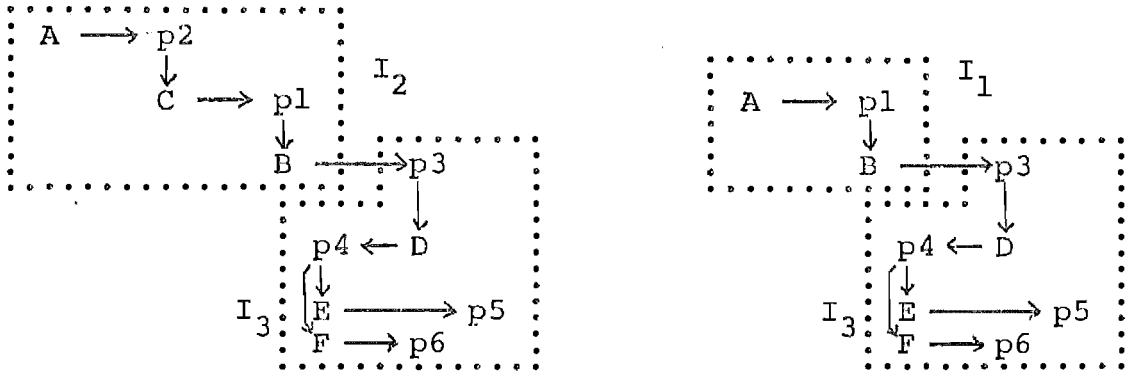


Figure 6.26 Solutions constructed from interchanges of 6.25

However, exceptions arise if common periods or clashes are involved. For example, figure 6.27(a) is a solution from interchange I_2 (if we assume that D does not clash with C), but if we attempt to create a solution from interchange I_1 and D clashes with A (6.27(b)), we find that D clashes with a fixed assignment in period p1. We shall term this result a self-clash.

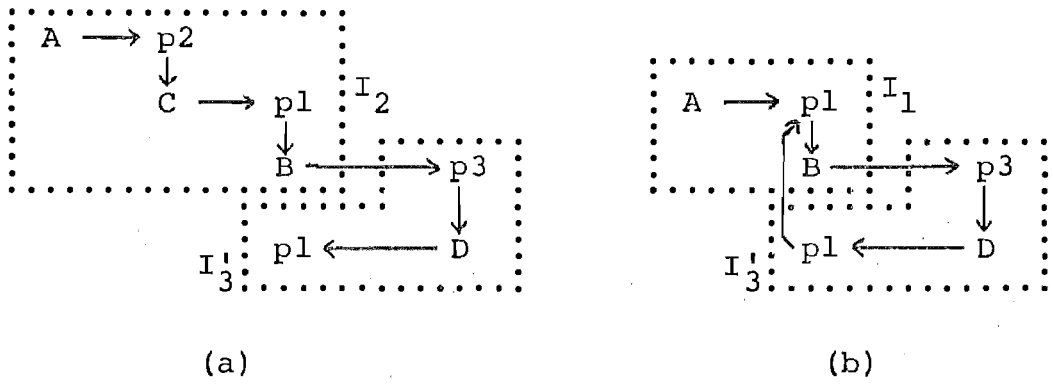
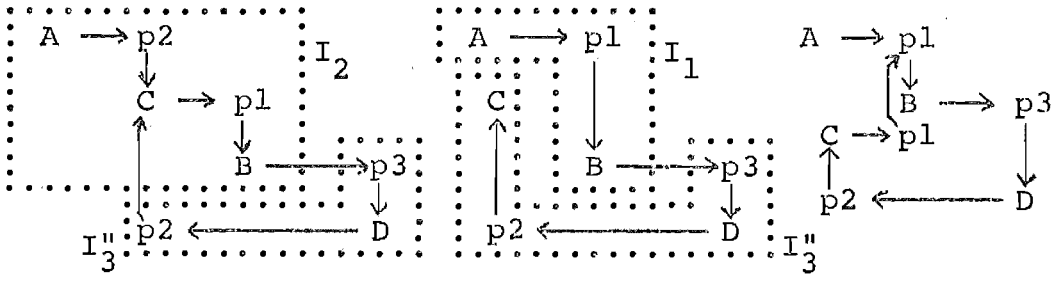


Figure 6.27 Self-clashing

Another exception arises when a clash occurs between a destination entry in the portion of the complete interchange following B and a source entry in partial interchange I_2

(figure 6.28(a)). D in p2 would displace C had it not already been displaced by A in p2. Again if we attempt to construct a solution from I_1 (6.28(b)), the result is not a solution since C is not re-entered. However we can use some of the source and destination entries in I_2 to extend this interchange (6.28(c)). In this case this results in a self-clash, but in others it can be a solution.



(a) (b) (c)

Figure 6.28 Clashing with Source Entry

In general, if $I_2 + I_3$ is a complete interchange, where I_2 is a partial interchange having one unsatisfied entry and I_1 is another partial interchange having the same unsatisfied entry as I_2, then $I_1 + I_3$ can be extended (by adding source and destination entries of I_2) to either a solution or a self-clash.

If I_2 has more than one unsatisfied entry, then I_3 becomes a set of interchanges, one for each unsatisfied requirement in I_2. If I_1 also has more than one unsatisfied entry, then the technique above can be applied if the unsatisfied entries of I_1 form a subset of the unsatisfied entries of I_2. For example, suppose $I_2 + I_3$ is the interchange shown in figure 6.29. I_2 displaces both B and D. If I_1 is a partial interchange which displaces just B, then a solution can be constructed from I_1 by using the right-hand interchange in I_3.

These observations suggest a simple procedure for pruning the search tree. If, on testing, an interchange I_2 is found to have a set of unsatisfied entries which is a superset of the unsatisfied entries for an interchange I_1 that has already been tested, then do not search from that interchange. If

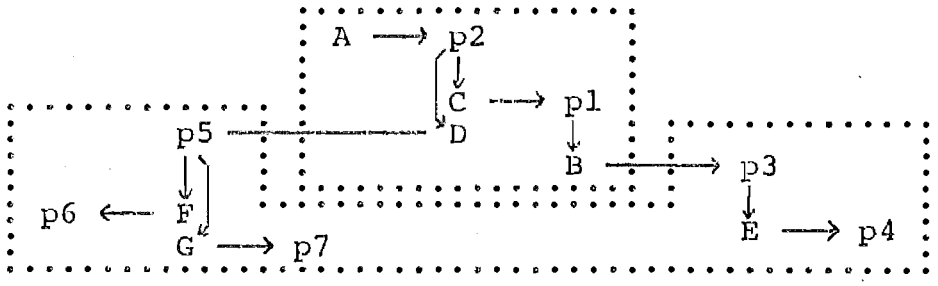


Figure 6.29 Solution from Interchange with Two Unsatisfied Entries

a solution can be found by searching from I_2, then either a solution or a self-clash will be found from I_1.

The difficulty with this procedure is in how to deal with a self-clash. There appears to be no simple technique to discover if there is a solution from I_2 which corresponds to the self-clash from I_1.

Consequently, the above procedure was tested as a heuristic in a program which merely ignored the self-clashes. The justification for doing this lies in the relatively small proportion of periods and assignments in the timetable that are affected by the interchange I_1 and hence the low probability of a self-clash. This is particularly so during the singles-loading stage.

6.12.1 Superset-Eliminating Algorithm

The algorithm uses the same stored search-tree structure as the best-node-expansion algorithm, but it is basically a simple breadth-first search. An interchange is stored as a node in the tree only if its unsatisfied entries do not form a superset of the unsatisfied entries of any other interchange stored in the tree. If its unsatisfied entries form a subset of those of any other stored interchange, then the stored interchange is deleted from the tree, together with its descendants.

1. Initialise: R ← requirement to be put in,
   NMAX ← 0, NODE ← 0, SET ← $\emptyset$.

2. P ← 1.

3. Test requirement R in period P. If P is forbidden for R, go to 12. Otherwise CLASH ← set of clashing assignments.

4. If SET $\cup$ CLASH = $\emptyset$, print out solution and exit.

5. N ← NMAX.

6. If N = 0, go to 11.

7. TRSET ← set of unsatisfied entries of interchange associated with node N.

8. If SET $\cup$ CLASH $\supseteq$ TRSET, go to 12 (do not create node on tree).

9. If SET $\cup$ CLASH $\subset$ TRSET, delete node N together with its descendants.

10. N ← N - 1, go to 6.

11. Create new node on tree:
    NMAX ← NMAX + 1
    UNEXP ← UNEXP $\cup$ {NMAX}
    PRD(NMAX) ← P
    PTR(NMAX) ← NODE

12. P ← P + 1; if $P \leqslant P_{max}$ go to 3.

13. NODE ← min N ε UNEXP.

14. Set timetable to state of NODE: SETT(NODE).
    R is set to new current requirement.

15. Delete NODE from set of unexpanded nodes:
    UNEXP ← UNEXP - {NODE}.

16. SET ← (set of unsatisfied entries in interchange represented by NODE) - {(R, source period of R)}.

17. Go to 2.

6.12.2 Application of Superset-Elimination to Timetabling

The major drawback of the program with superset-elimination was lack of speed. With the complex procedure involved in setting the timetable to arbitrary nodes in the tree, together with the search of the tree required for each period test, the program was much slower than the simple depth-first search, even when techniques were introduced to accelerate the search of the tree.

Another problem was the large number of possible sets of unsatisfied entries. Few interchanges with two or more unsatisfied entries were rejected because the chance of finding that the entries of one such interchange form a superset of those of another is reduced as the numbers in the sets are increased. Again, this remained a problem even after techniques were introduced to increase the chance of finding supersets amongst the nodes.

To increase the speed it may be necessary to replace the breadth-first search by a depth-first one in which the search tree is not stored. The depth-first mothod searches the tree in a manner which minimises the number of load/unload operations required on the timetable.

However, application of superset elimination to depth-first searching is fraught with difficulties since, unlike the breadth breadth-first method, interchanges are not examined in the order of increasing size. Problems are also introduced by the need to have a depth limit in depth-first searching. No satisfactory way of overcoming these obstacles was found.

6.13 A Suggested Method for Tree Searching Singles

None of the algorithms described so far are able to handle the 'last assignment of a class' problem satisfactorily. The superset-eliminating algorithm is handicapped by lack of speed, while the others make no special effort to aim for the last empty period of a class in an attempt to fill it.

To complete this discussion of tree-searching algorithms, a suggestion will be made on how this problem might be tackled.

The method to be described has not been programmed or tested in any way. It is intended as a starting point for further work.

The principle of the method is as follows. A 'higher-level' tree search is carried out, in which only the empty periods for classes are examined. When this tree search finds an apparent 'solution', the ordinary depth-first tree search is used to determine whether this represents a true solution. If so, the solution is printed out. Otherwise the 'higher-level' search is continued.

Typically, for each empty period for a class, there is a choice of only about 4 to 7 single-teacher single-class assignments and about 3 to 5 blocks that can be entered into the empty period. This gives a relatively low branching factor for the high-level search.

As in all of the other tree-search based algorithms, the initial requirement to be entered is specified by the time-tabler. In general, this requirement will involve several classes. One of these classes must be chosen for analysis by the high-level search. An appropriate heuristic is to choose the class (or item in general) with the minimum number of empty periods.

Having chosen such an item, the high-level search examines the clashes of each of the requirements of that item in each of the empty periods of that item. For each possibility, a tree node is generated to represent the set of clashes so obtained. These nodes are linked back to the root of the tree.

For clarification, consider the example in figure 6.30. Assignment A, consisting of items a, b, and c, is to be entered into the five-period timetable shown. The item b has only one empty period, so it is chosen for analysis. Its empty period is period 4. The requirements involving item b are A, F, D, H, and J. When each of these is entered into period 4, it displaces assignments as shown in figure 6.31(a). The tree shown in 6.31(b) is generated.

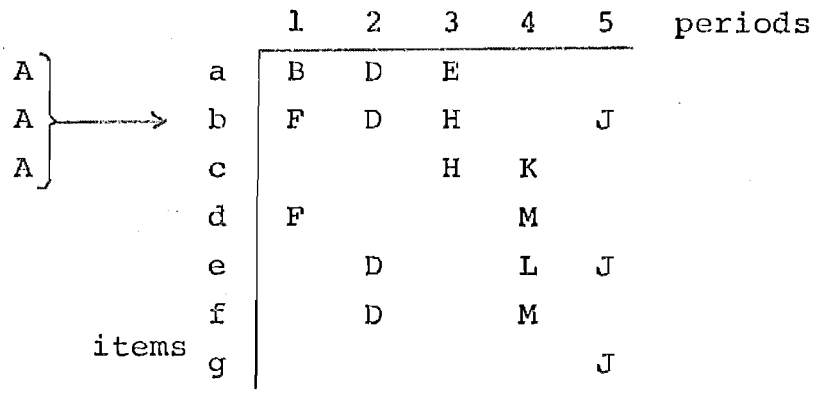


Figure 6.30 Example Problem for High-Level Tree Generation

(a)

Requirement involving item b	Clashes in period 4
A	K
F	M
D	L,M
H	K
J	L

(b) Tree:

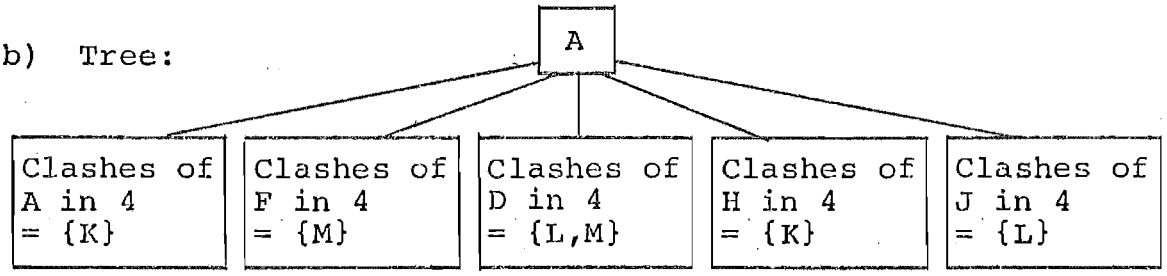


Figure 6.31 High-Level Tree Generation

One of the tree nodes is then chosen for further expansion. A heuristic can be used to determine the 'easiest' node. The requirement of one of the clashes in the node becomes the new current requirement. The clash is removed and its source period is disallowed so that it will not be re-entered. The above procedure is repeated for the new requirement and further nodes are added to the tree.

In the example of figure 6.31(b) the third node from the left may be chosen for further expansion. This node contains clashes L and M. Suppose clash L is chosen as the new current requirement. L is removed from period 4 of the timetable, and this period is disallowed for L. New nodes are then added for L (figure 6.32).

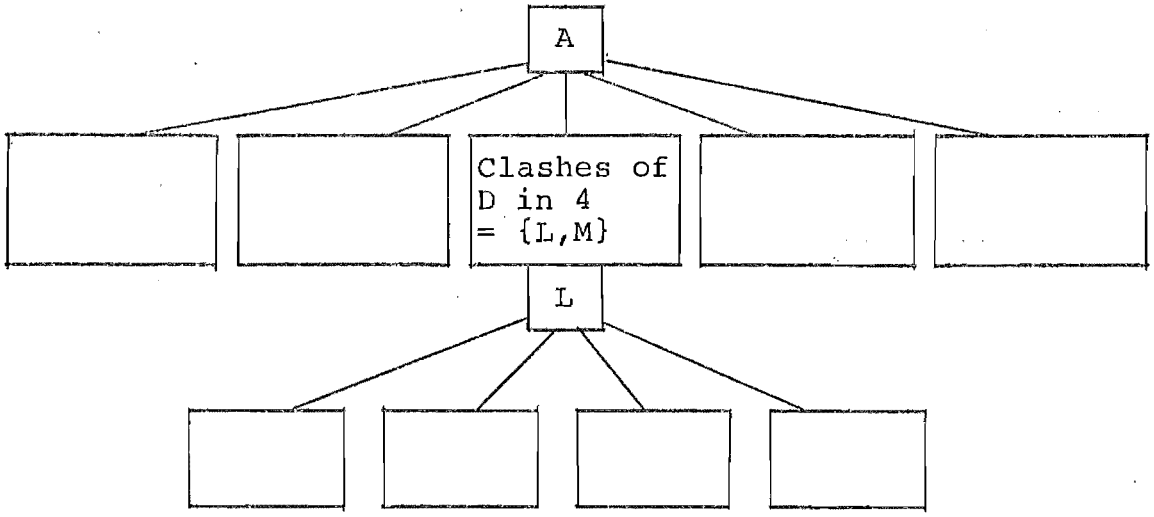


Figure 6.32

When a node which contains no clashes is to be expanded, the path from the node to the root is retraced in a search for a node containing unresolved clashes. If such a node is found, the requirement of one of the unresolved clashes becomes the new current requirement. For example, if one of the nodes below L has no clashes (figure 6.33) then on retracing towards the root, M is found as an unresolved clash and hence is chosen as the next current requirement.

If no node with unresolved clashes is found during the retrace, then a potential solution is indicated. The ordinary tree search can then be used to 'link up' the nodes of the high-level tree from the root to the no-clash node (figure 6.34).

At each linking step, the clashes in the successor node are removed and the chosen requirement from the predecessor is entered by tree searching. If the tree search is successful, the next pair of nodes are linked in the same way. If the

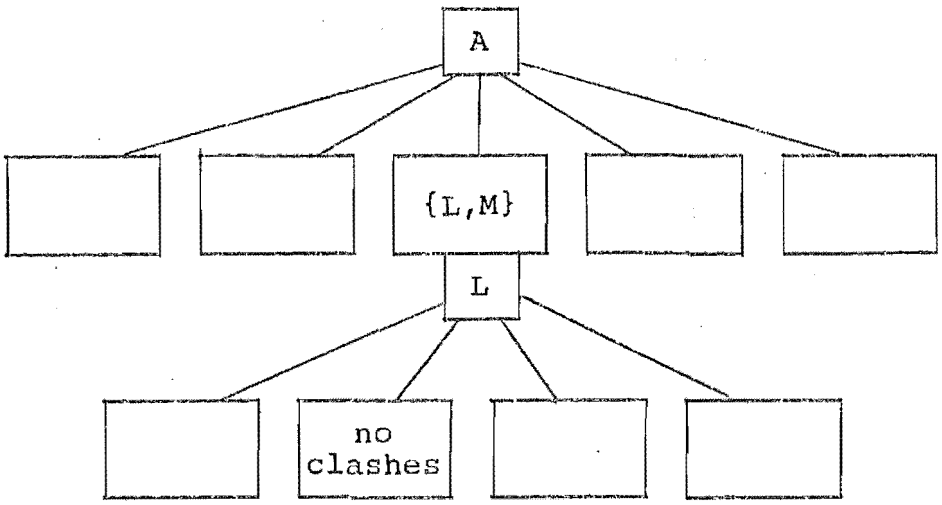


Figure 6.33

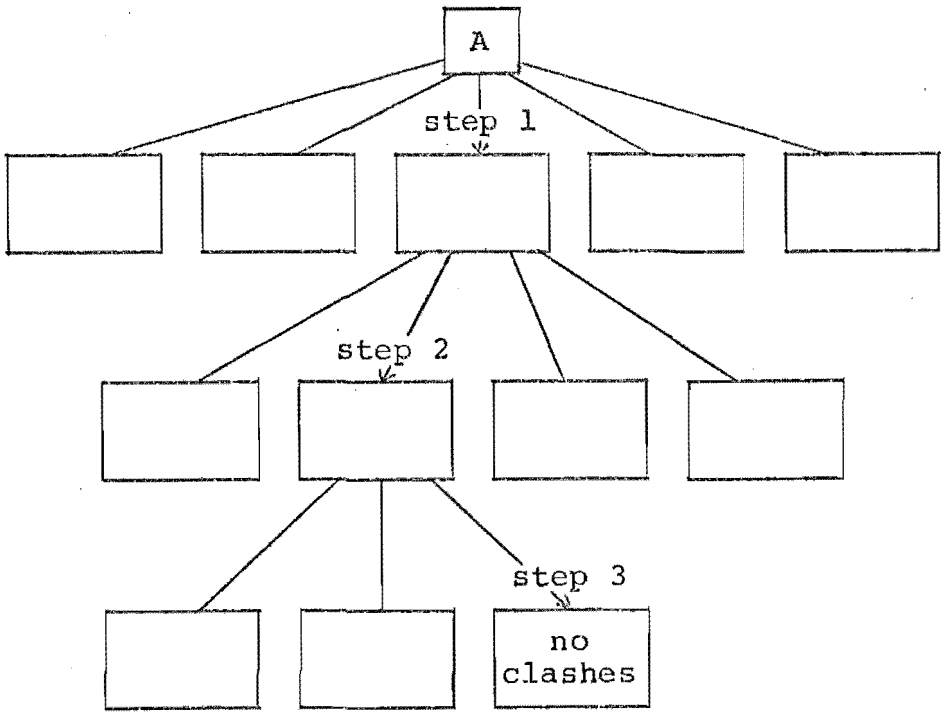


Figure 6.34

tree search is not successful, then the successor node and all of its successors are deleted and the high-level search is continued.

The overall aim of this method is to find a 'deep' stack by performing a sequence of relatively 'shallow' searches rather than by attempting one 'deep' search. Because of the exponential increase of search time with increasing stack depth, this method promises significant reductions in overall execution times.

However the method assumes that there can be no interaction between any pair of branches in the high-level tree. Two branches interact when it is possible that a 'low-level' tree linking the nodes of one branch has source and destination nodes nodes which clash with the nodes of the tree forming the link for the other branch. If this situation occurs then the method will not be exhaustive. In the singles stage of practical timetabling, the number of assignments in the timetable is large and the probability of such interaction occurring is thus relatively low. The application of the method must therefore be restricted to singles problems and even then it is possible for the occasional solution to be missed.

6.14 Techniques to Aid Interaction with the Tree Search

6.14.1 Background Tree Searching

In normal interactive use of the tree search, the timetabler enters the requirement and the search depth, and then must wait until either a solution is found or until he decides that searching has continued for long enough. If standard methods of programming are employed, the timetabler cannot execute any other commands while the tree search is in operation.

However it is possible to apply a multiprogramming technique to tree searching. The tree search routine can be run in the 'background' while display commands are executed in the 'foreground'. This is accomplished in the following manner.

After the timetabler has entered the parameters for the search, the computer returns to the command mode but instead of waiting in a pause state until command characters are entered, it begins the tree search. When the timetabler enters the characters, the tree search is interrupted. Before the interrupt is serviced, the tree search routine backtracks to the begin-

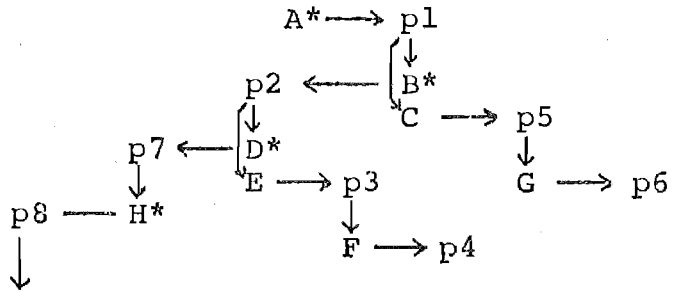
ning of the stack and thereby restores the timetable to its original state. The stack is saved. The command that has been requested is then executed. When the execution of the command is complete, the saved stack is used to restore the timetable to the state at the time of the interrupt. The tree search then continues.

This facility allows the timetabler to look for possible requirement alterations or constraint relaxations while the tree search attempts to find a solution to a difficult problem. The overall time to construct a timetable is thereby reduced. The timetabler can also request a listing of the stack without interfering with the operation of the tree search. Alterations to the timetable or requirements while the search is in progress are prevented by means of program switches.

6.14.2 Flagging of Requirements

When the tree search fails to find a solution, it would be beneficial for the timetabler to know whether it came close to a solution at any point, and whether the changing of any requirement will enable a solution to be found. Some information to this effect can be obtained by modifying the tree search routine to flag requirements in the manner to be described and to print out the flagged requirements upon request from the timetabler.

Requirements are flagged whenever the depth limit is reached and there is not more than one unsatisfied requirement at the time. The requirements that are flagged are those that are on the chain leading to the unsatisfied requirement (figure 6.35). If such a flagged requirement could be entered directly without clashing then a solution would be found. There may in fact be a non-clashing period for the requirement, but distribution constraints may have prevented the use of that non-clashing period. Knowledge of such information will help the timetabler to make a change or relaxation which is appropriate to the problem under investigation and which will assist in finding a solution to the problem. In particular the presence of non-clashing periods for a flagged requirement can be indicated directly on the listing of the flagged



* These requirements are flagged

I* — unsatisfied. Depth limit was reached at this point. If a free period had existed for any one of A,B,D, H or I, then a solution would have been found.

Figure 6.35 Flagged Requirements

requirements. A simple relaxation of a distribution constraint may be all that is necessary to allow such a requirement to be entered.

6.15 Fitting 'Difficult' Blocks by Tree Searching

Although the modified tree-search algorithm of section 6.9 is capable of moving many assignments to allow a new block to be entered, situations arise in computer-aided timetabling in which even prolonged tree searching fails to find a solution. When this happens, some of the school's originally imposed constraints have to be relaxed or the assignment of teachers to a block has the be changed, as it is unlikely that any other method will find the rearrangement of the timetable necessary.

The problem then occurs of deciding which constraint or constraints to relax. The constraint relaxation must be acceptable to the school and must also allow a solution to be found to the current problem. The simple technique of relaxing one constraint then repeating the tree search on a trial basis is wasteful of time as the one constraint relaxation usually makes little difference to the difficulty of the problem.

A better approach is to relax all constraints for which violations will be accepted by the school, before attempting to enter the outstanding blocks. After all of the blocks have been fitted, the relaxed constraints are reintroduced. This technique reduces the total tree searching time and ensures a good quality final timetable. The following section illus-

trates the application of this technique to the construction of a timetable for Riccarton High School in February 1974.

6.15.1 Constraint Relaxing and Reintroduction - an Example

Figure 6.36 shows the state of the timetable after most of the blocks had been entered. The display is in the class-period format of figure 4.4. The fitting of the blocks had up to this point been largely carried out by a heuristic routine which was similar to that of Appleby et al (1961) but which called up the tree-search routine when the value of P-N for a requirement became negative. When the timetable had reached the state shown in figure 6.36, the tree-search routine could not enter the last of four assignments coded '#F'. Only a few blocks had been flagged by the technique of section 6.14.2. These blocks had no free periods and clashed with many of the other blocks in the timetable, so no applicable constraint relaxation was obvious. A requirement display for #F in this timetable is shown in figure 6.37. It was absolutely essential that all of the blocks were in the timetable and something therefore had to be done to enable the #F as well as four outstanding #I's to be entered.

The first step taken was to relax most of the low-priority constraints on the blocks in the timetable. Period unavailabilities for teachers were removed, doubles were split into singles, and a tie forcing all assignments +I and [3 to occur simultaneously (the tied pair was represented by +J) was removed. When these constraint relaxations were made, the timetable was left effectively unchanged. For example a double in the timetable was replaced by two adjacent singles. Thus no violations were introduced.

After these constraints were relaxed, the tree search found a solution for #F in 6 seconds. This solution separated a double @F which was split in the constraint relaxation, but did not violate any of the other relaxed constraints.

Three out of the four outstanding #I assignments (involving classes 3I, 3J, 3K) were fitted successfully by tree searching although the third took a total time of 13½ minutes and re-

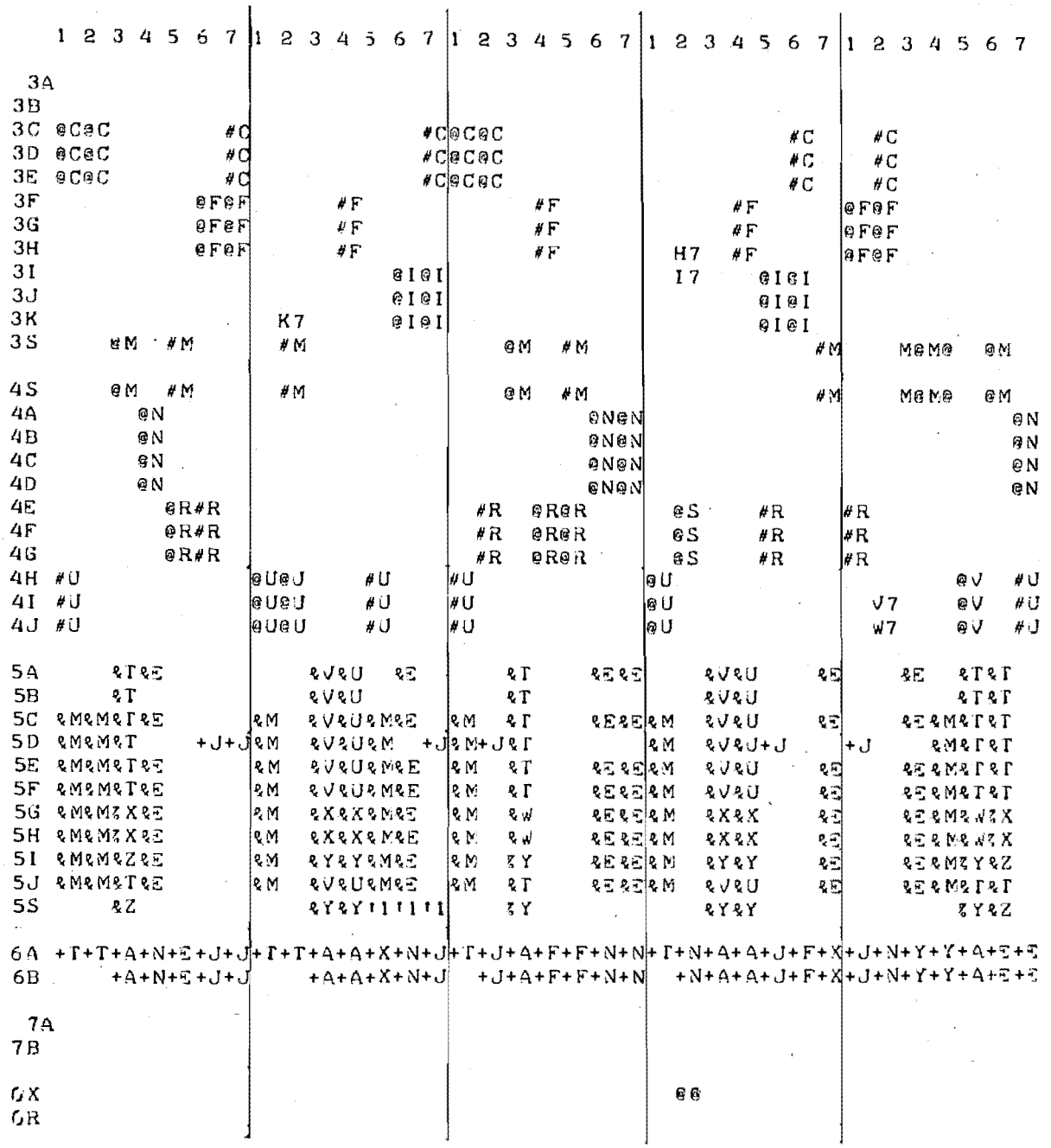


Figure 6.36 Timetable with One #F and 4 #I's Outstanding

Figure 6.37 Clashes of #F in Timetable 6.36

quired the large interchange shown in figure 6.22. The fourth could not be entered. The timetable at this point is shown in figure 6.38 and a requirement display of #I in figure 6.39.

Further constraint relaxations were necessary. The distribution constraints on the '#' requirements (representing Technical Drawing/Typing) were relaxed to allow one day with two lessons. Art preassignments were removed. The preassigned #U was allowed in all periods. The distribution constraint on &E was eased. A 'theory' lesson for the woodwork and metalwork classes in @R was created (in other words the woodwork and metalwork rooms were removed from one assignment of @R).

With these changes in effect, the final #I was entered after 18½ minutes of tree searching. Now that all of the 'difficult' blocks were in the timetable, the process of constraint reintroduction could be started. All but one of the art preassignments were reloaded, non-available periods for teachers were reintroduced, the theory class was changed back to practical, and double periods were re-formed. During this process any assignment violating a reintroduced constraint was removed and was re-entered by tree searching with the constraint applied. Usually this was successful and an improved timetable resulted. However in a few cases the tree search failed. For example, the fitting of assignments with constraints relaxed had resulted in the placing of two assignments of #I on the same day and likewise for #F. Tightening the distribution constraints on these to allow only one per day resulted in one of each assignment being displaced. Tree searching of both failed. The arrangement of the #F and #I assignments was therefore left as it was. The failure of the tree search confirmed that the relaxed distribution constraint was necessary for completion of the timetable.

In addition to this, it was not possible to realign one '+I' with a '[3', and a double '+T' had to remain as two separated singles. However the effect of reintroducing the constraints improved the quality of the timetable by greatly reducing the number of violations and it confirmed that the constraint violations that remained were necessary. The resulting time-

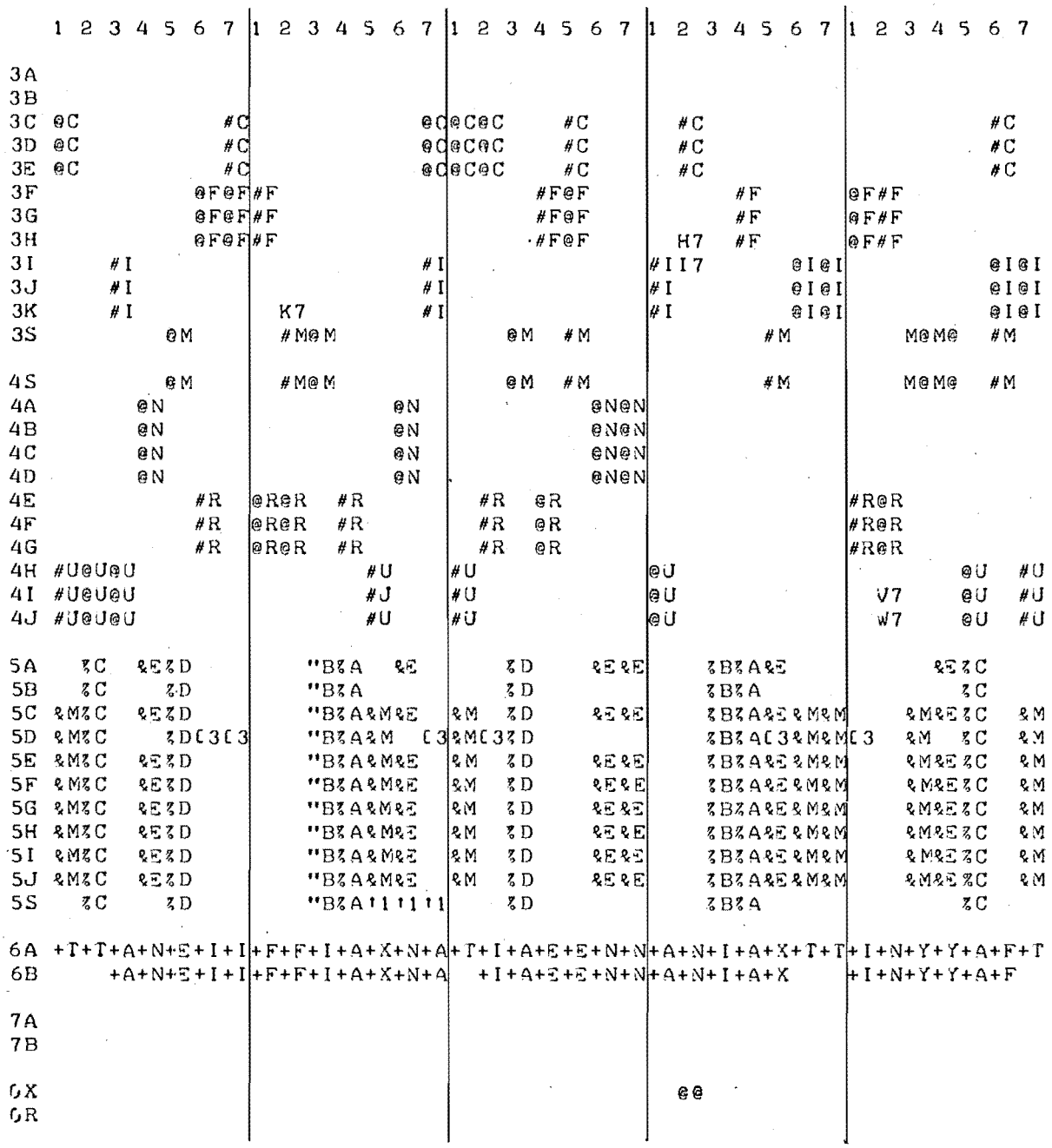


Figure 6.38 Timetable with One #I Outstanding

Figure 6.39 Clashes of #I in Timetable 6.38

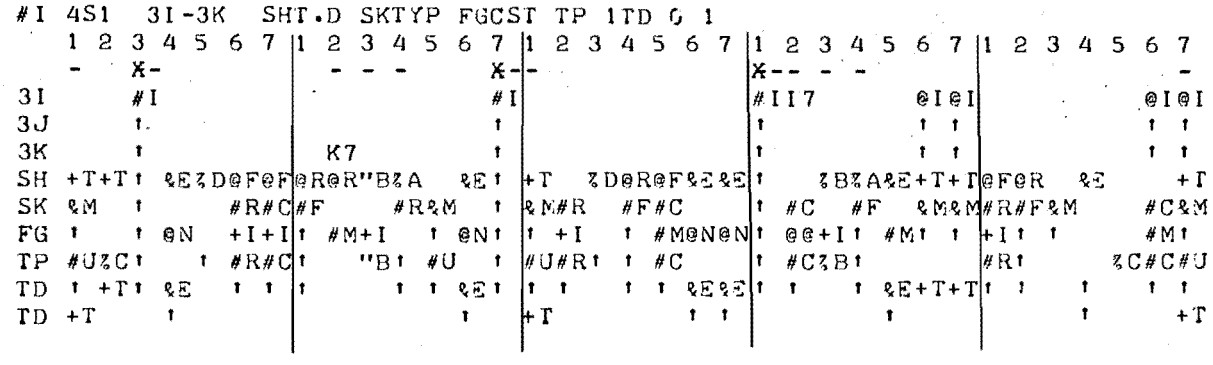


table is shown in figure 6.40. All remaining blocks were then fitted, as well as a few smaller assignments which became tight. Then all the remaining relaxed constraints were re-introduced. No timetable assignments were displaced in this final phase. The timetable, now ready for the loading of singles, is shown in figure 6.41. The constraint violations that remained were:-

a) One double for each '@' block (woodwork/metalwork) had become separated into two singles. However as the second double in each case was only desirable rather than essential, no attempt was made to re-form the double.

b) #F and #I each had two lessons on one day.

c) One +I assignment was not aligned with an assignment of [3.

d) There was only one double of +T instead of two, as was originally required.

6.15.2 Constraint Relaxing and Reintroduction - Conclusion

The foregoing example demonstrated a technique by means of which the process of introducing constraint violations and making compromises can be controlled. If the technique is not followed, much time is liable to be wasted in trying alternative changes, and the resulting timetable will have far more compromises and undesirable features than is necessary. The essential features of the method are as follows:-

a) All constraints are included in the computer data at the start of construction. This ensures that the final timetable will at least satisfy a reasonable proportion of them. If they are not initially included, the computer will not attempt to satisfy them and violations will therefore be numerous.

b) By relaxing all non-essential constraints, we reduce tree searching time to a minimum since flexibility is greatly increased. The rule for choosing constraints to be relaxed should be that a timetable which violates all relaxed constraints can still be used, although it would be a very poor timetable.

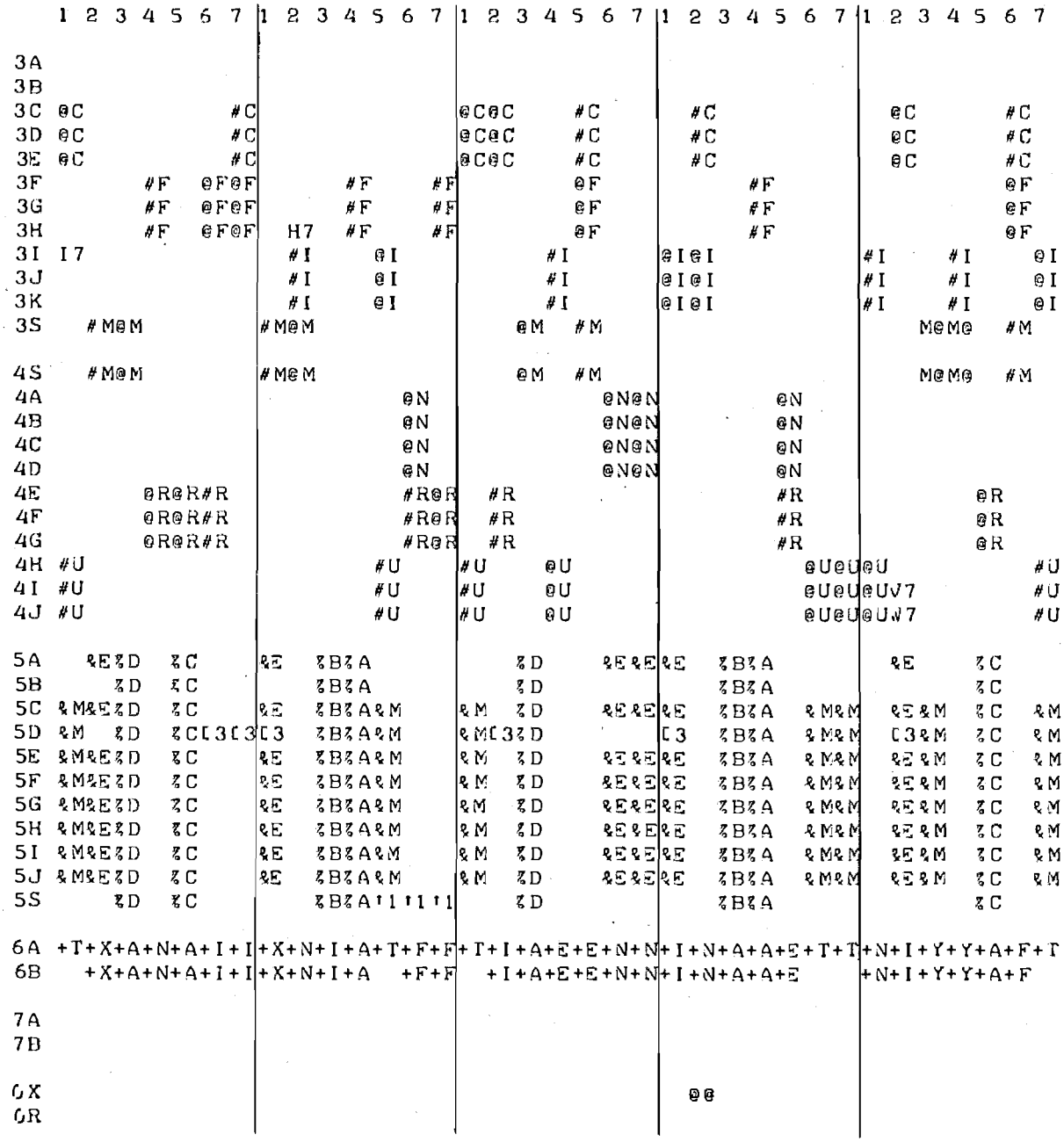


Figure 6.40 Timetable with All Major Blocks Fitted

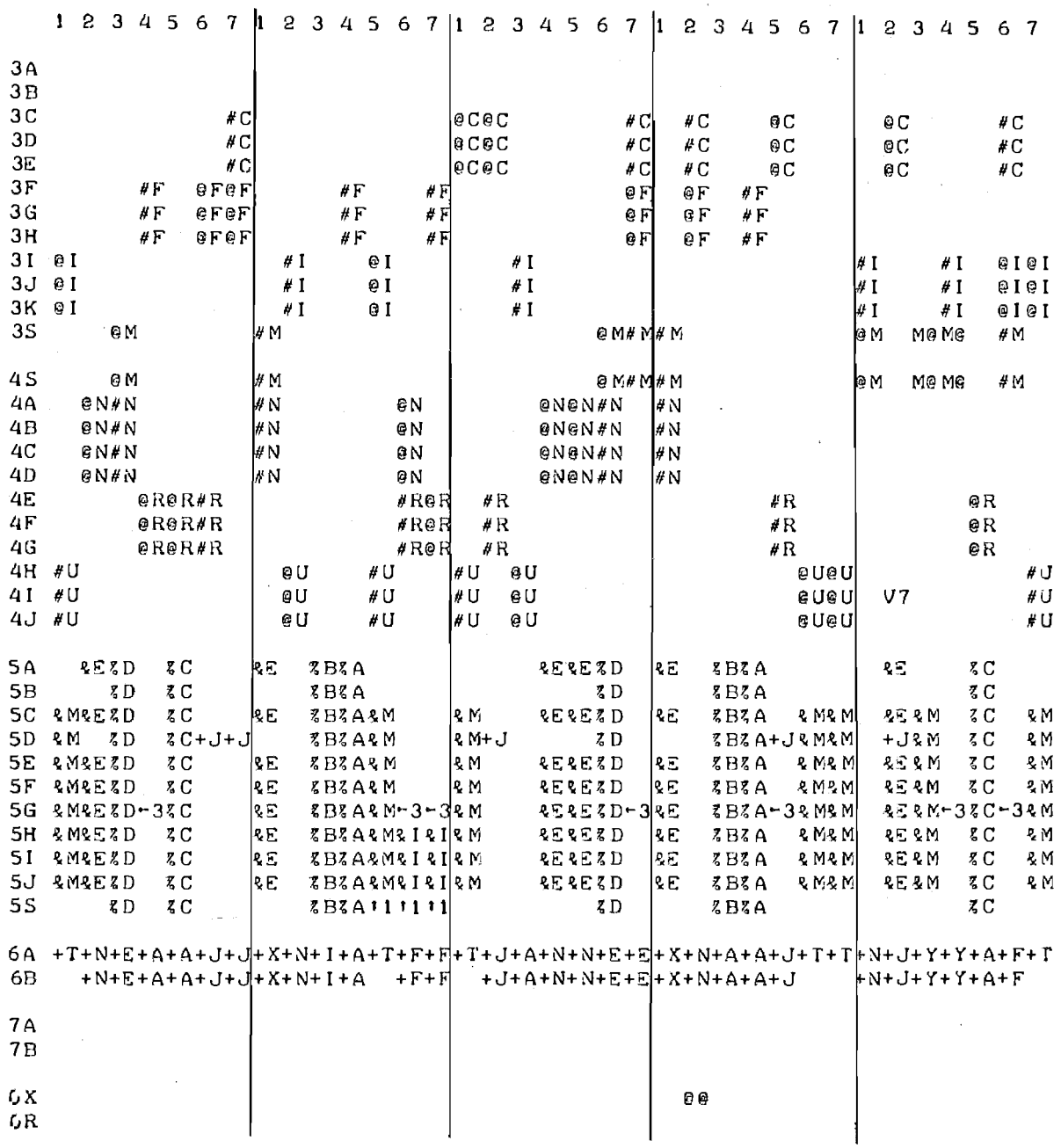


Figure 6.41 Timetable after All Constraints Reintroduced

c) The process of tree searching the remaining blocks after relaxing the constraints in fact produces violations to only a small proportion of the relaxed constraints. The tree-search routine will violate a constraint only if a solution can be found by doing so.

d) Constraints are reintroduced one by one, in order of decreasing desirability. If any violations of a reintroduced constraint have occurred, the offending assignment or assignments will be displaced from the timetable. An attempt is made to re-enter the displaced assignment(s) by tree searching. If this attempt succeeds, the violation no longer exists. If it fails, the timetable is restored to its original state, with the constraint violated. The original choice of constraint relaxation will ensure that the restored timetable will be acceptable to the school. The benefits of constraint reintroduction are:-
(1) It further reduces the number of constraint violations in the final result.
(2) It prevents new violations appearing during the later stages of constraint reintroduction or in the following construction activity. The order of reintroduction from greatest to least desirability ensures that, if two constraints conflict, the more desirable one will be satisfied.
(3) If the tree search fails, the constraint violation that therefore remains is confirmed as being necessary for completion of the timetable. This information is important to the timetabler who has to justify the violation to the school staff affected.

If the tree search fails even after all possible constraints have been relaxed, then it is unlikely that any other method will succeed in finding a solution with the data presented. The timetabler therefore is effectively told that the data is impossible and that it is necessary to spend time in examining the block structure.

6.16 Other Special Applications of Tree Searching

6.16.1 Improving Distribution

At various points in timetable construction and particularly at the end, it will be found that it is necessary to move assignments to improve distribution or to satisfy ill-defined constraints. In particular, it may be necessary to improve one of the following:-

a) Distribution of lessons in a particular subject for a class.

b) Distribution of free periods for a teacher.

c) Grouping of teaching periods for a part-time teacher.

The technique for carrying out such improvements is simple and effective:-

a) Remove a badly placed assignment.

b) Disallow the period from which the assignment came, to prevent the tree-search routine attempting to re-enter it into that period.

c) Disallow any other periods which are badly placed for the assignment from the point of view of distribution.

d) Enter the assignment by tree searching.

e) Depending on the circumstances, the entered assignment may be fixed or other periods may be disallowed to prevent the new arrangement from being destroyed by subsequent alterations. However it must be kept in mind that this reduces flexibility.

As an example, suppose that at the end of construction, the arrangement of assignments on one day for a part-time teacher is as shown in figure 6.42. The assignments are badly grouped. We wish to move C3 closer to #I and @B, but other assignments clash with C3 in all periods except 7. To do this, C3 is removed and the afternoon periods are disallowed for this assignment. The afternoon periods are also disallowed for #I and @B, unless the latter are large blocks in which case it is

1	2	3	4	5	6	7
	#I	@B				C3

Figure 6.42 Initial allocation arrangement for part-time teacher

	1	2	3	4	5	6	7
		#I	@B				
#I:	*	#I	*	*	-	-	-
@B:	*	*	@B	*	-	-	-
C3:	*	*	*	*	-	-	-

* = clash
\- = disallowed

Figure 6.43 C3 Removed, Afternoon Periods Disallowed

	1	2	3	4	5	6	7
		#I	@B	C3			
#I:	*	#I	*	*	-	-	-
@B:	*	*	@B	*	-	-	-
C3:	*	*	*	C3	-	-	-

Figure 6.44 C3 Re-entered by Tree Searching

unnecessary to do so (figure 6.43). Tree searching now enters C3 into period 4 (figure 6.44). The disallowing of the afternoon periods may now be retained. This will allow future alterations to change the positions of #I, @B, and C3 within the morning periods where they will remain relatively well-grouped. A more severe restraint may be made by fixing #I in period 2, @B in 3, and C3 in 4. This will ensure that the grouping remains, but will restrict any alteration that attempts to move any one of them.

6.16.2 Handling Blocks of Complex Structure

The school's block timetable may include blocks in which one or more subjects are shared among several teachers or in which one or more groups take more than one subject. Figure 6.45 is an example of such a block. Four periods are required for this block. In two of the four, Cd takes Typing. In the other two, Ef takes typing. Also French is taken by Gh in two periods and by Ij in the other two. It does not matter whether Ef teaches at the same time as Gh or Ij, but the French, the

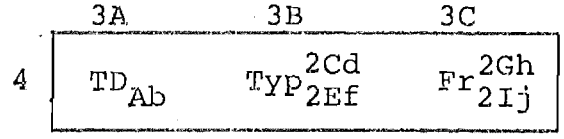


Figure 6.45 Block in which Two Subjects are Shared

Typing and the Technical Drawing lessons must occur simultaneously in the same four periods.

If only one subject is shared, then one requirement is defined for each teacher sharing this subject. The requirements defined differ only in the teacher taking the shared subject, and they all form one distribution group. The complex structured block can then be handled by the system.

However the special flexibility inherent in a block with two or more shared subjects poses a problem. Separate requirements, one for each teacher taking a shared subject, must be defined in order to take advantage of the flexibility. Effectively the block must be split into components as illustrated in figure 6.46.

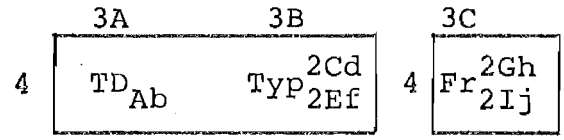


Figure 6.46 Shared-Subject Block Split into Components

Requirements can be defined for each component of the block in the manner described above. The problem is then to ensure that the components remain aligned in four periods of the timetable and do not become separated. If they do become separated, other assignments entered later into the timetable may prevent the tree search from restoring the alignment.

The solution to this problem is as follows. One dummy class is created for each component of the block and is assigned to that component. In the above example, 3X may be assigned to the TD-Typ component, and 3Y to the Fr component. A new requirement is created which contains just the dummy classes 3X and

3Y. The number of periods in this requirement is set to the difference between the number of periods in the week and the number of periods required by the block. When all assignments from this new requirement are allocated to the timetable, they will have the effect of 'forcing' the block components into alignment by clashing generated by 3X and 3Y (figure 6.47).

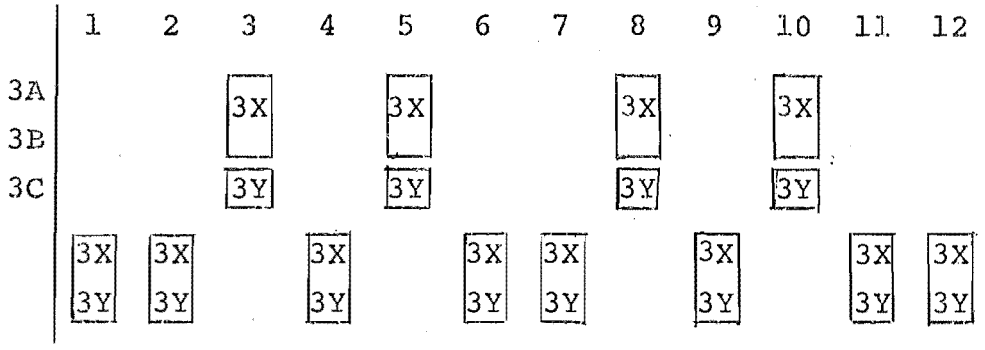


Figure 6.47 Timetable Containing Block Components

If the tree-search routine moves one of the block components to a period not containing an assignment of the block, one of the 'dummy' assignments will be displaced. The tree search must then replace this assignment. In most periods an identical 'dummy' assignment will already be present and will therefore be avoided by the tree search. The only periods available will be those in which other assignments for the block are present. The tree search is thus 'forced' to move an associated component of the block to restore the alignment (figure 6.48).

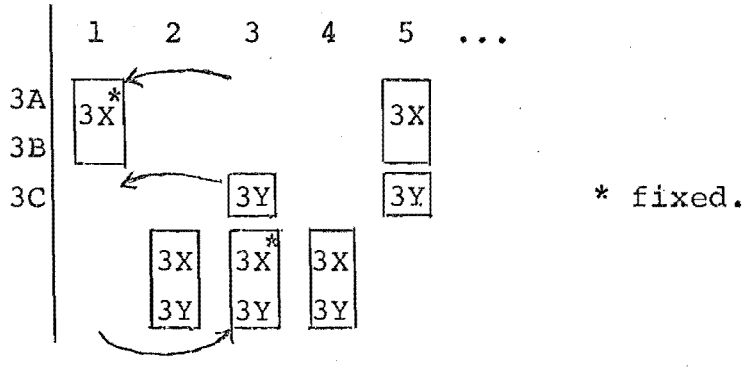


Figure 6.48 Effect of Moving Block Component

It may move the 3C component of the block into period 1, or it may exchange this with another 3C component containing a different teacher. Alternatively it may enter the 3C component into, say, period 2, displacing a dummy assignment which in turn causes block components from another period to align with those in periods 1 and 2.

6.17 Conclusion

A tree-searching routine has been found to be absolutely essential for the success of the interactive aid. The modified algorithm of section 6.9 gives the best overall performance, but a technique similar to that described in section 6.12 is required to improve its ability to manipulate singles. The tree search is more than just a routine for entering assignments into the timetable. It forms the basis of an interactive technique to save time and to minimise undesirable features in the final result when the timetable problem is difficult. It enables ill-defined constraints to be satisfied and it provides a means for handling complex block structures. It has thus shown itself to be highly versatile as a means by which the timetabler can solve a variety of timetabling subproblems with a minimum of tedium and effort.

CHAPTER 7

PRACTICAL APPLICATION

7.1 Introduction

In this concluding chapter, the results of the use of the interactive system for constructing timetables for Riccarton High School are given to show the success that has been achieved. Times are presented to show the improvement over the manual method, and an attempt is made to give an indication of the quality of the result that has been achieved.

The second main section in this chapter considers the future of the system. The ultimate goal is to make the system available to all schools. However as many aspects of the process of making the system available depend on external circumstances such as the source of finance, this section considers basic requirements only. These requirements include hardware, further program development, user training and maintenance.

7.2 Results of Timetable Construction Runs

7.2.1 Times

The interactive timetabling aid implemented on the EAI 640 at the University of Canterbury was used at various stages of its development for constructing timetables for Riccarton High School. Altogether four timetables were constructed, two for the school year 1973 and two for 1974. Two timetables were constructed each year to keep in line with the School's policy of making one timetable in November for use during the first few weeks of the first term in February while the final version is prepared from more up-to-date information.

Times for the construction runs in February 1973, November 1973, and February 1974 are listed in table 7.1. The total times include all of the time spent on the timetable, both on and off the computer, from the receipt of the block timetable to the printing of the finished result. Time spent by the heads of the science departments at the school in allocating science labs is not included. The second line of the table gives the total time the computer was used for timetabling, while the third line gives the time from the point at which the first assign-

	Feb 1973	Nov 1973	Feb 1974
Total Time (hours)	43½	33¼	25¾
Time Spent Interacting with Computer	not recorded	24	21
Time from Empty Timetable to Full Timetable	28	12	9¾
Times for Each Stage			
Data Preparation & Checking	9	14	5½
Loading Blocks	20	8	7¼
Loading Singles	8	4	2½
Improving Distribution	2	2¼	5
Assigning Rooms & Printing	4½	5	5½

Table 7.1 Times for Timetable Construction Runs

ment was entered into the timetable to the point at which the timetable was first complete. Times for each major timetabling stage are also shown.

These times can be interpreted directly as man-hours. Although both the author and the school's timetabler were present during the computer sessions, the work load throughout could be handled by one person. On this basis, it can be seen that all of the total times represent a significant reduction on the 160 man-hours quoted in chapter 2 for manual timetable construction. It can also be seen that the time for the timetable construction itself in November 1973 and February 1974 is less than half of the overall time. Unlike manual timetabling, 'peripheral' activities in interactive timetabling occupy at least as much time as does the construction activity itself.

7.2.2 Quality of Result

There is no objective way of measuring the quality of a school timetable since many human factors are involved. However it was the opinion of the administration of Riccarton High School and of the School's timetabler that the timetables produced by the interactive system were at least as satisfactory as earlier manually-produced timetables. During the period of the time-

tabling project, the School had demanded greater flexibility and complexity in its course options and has thereby made the timetabling problem more difficult.

From a more technical point of view, the performance of the interactive aid can be assessed in terms of faults in the final result, such as poor distribution, assignments placed in the wrong periods, and pairs of single-period lessons forming unwanted doubles. These can be considered as violations of the originally imposed constraints and they can arise in three different ways:-

a) The original constraint included in the computer data had to be relaxed during construction to allow progress to be made.

b) The original constraint was not included in the computer data (because it was ill-defined) and could not be satisfied during construction within a reasonable time limit.

c) The original constraint was not included in the computer data because it was inadvertently omitted, and it was overlooked during manual checking.

The number of faults that arise depends also on the difficulty of the timetable. However the true quality of the timetable depends on the extent to which the faults can be tolerated. One intolerable fault is worse than many faults that cause only minor inconvenience. The interactive aid virtually allows the timetabler to choose the faults that will be tolerable in the final result. This is shown by the fact that in the February 1974 timetable there were a large number of faults of type (a) and (b), yet the worst fault (to the school) was one of type (c).

7.3 Implementation of the System

There are three possibilities for implementing the interactive timetabling system so that it is available for use by a number of schools:-

a) The system is implemented on a school's own mini-computer.

b) The system is implemented on a central time-shared computer and terminals are available in schools.

c) The system is implemented on a central computer which is physically accessible to schools of a city.

The first two alternatives provide the school with the advantage of freedom to construct timetables in its own time and of availability of computing power within the school itself. The timetabler is able to consult other staff members while constructing the timetable to discuss the various 'human' decisions that must be made. He can use the computer at virtually any time for any length of time.

The third alternative is more restrictive. The timetabler will be required to travel from the school to the computer and a roster system will be necessary to allow, say, a two hour session for each school per day. This mode of implementation is nevertheless not impractical and has the advantage over the first two that the overall hardware costs are lower. It could be used in the 'pilot' stages of setting up an interactive timetabling system before computers or terminals are widely available in schools.

7.3.1 Hardware

Having chosen an implementation, one must then consider the choice of hardware. Since nearly all school timetabling takes place during November to February of each year, a computer dedicated to timetabling would be largely idle for the rest of the year. For economic reasons, a computer used for interactive timetabling must be used for other purposes as well as timetabling. The specifications of the computer will be dictated as much by these other applications as by timetabling. Hence in what follows only minimum requirements will be given.

The present implementation of the system on an EAI 640 shows that a computer for interactive timetabling does not have to be large. A 16K core of 16-bit words is sufficient for the data for a medium sized school together with the program, written in Assembler and organised into three overlays -- two

for timetable construction and one for classroom assigning and printout. The number of overlays can readily be increased to allow data for larger schools to be fitted into the same core space. Disk space required for the programs in their present form is 18K words and each stored timetable takes 4½K words. A 64K disk would be adequate for storing programs and data. The instruction sets of most mini-computers are adequate for the timetabling system as there are no floating-point operations and the use of integer multiply and divide is restricted mainly to address calculation.

For any interactive system a display device is essential. The full benefits of interaction can be realised only if the display device has a quick response. Interaction using a 10 character per second teleprinter would be tediously slow and difficult although not impossible. More suitable devices would be either a CRT display scope or a fast typing or printing device.

A display device that is well suited to timetabling is the storage CRT display. The storage tube allows a large quantity of information to be displayed without flicker and points on the screen are individually addressable. Timetable displays may include horizontal and vertical lines and characters may be positioned anywhere on the screen. However the chief disadvantage of the storage display is that portions of a display on the screen cannot be selectively erased. Instead the whole screen must be erased and the display redrawn — a process taking more than a second.

A refresh display device of the type used for computer-aided design is also suitable for timetabling and has the added advantage of selective erasure. However it is too expensive to be considered as part of a school computer system.

In common use today is the VDU, which is a CRT display device employing a raster scan and containing an internal character generator. This device allows selective erasure and is relatively inexpensive, but it is somewhat restricted in its information capacity. The maximum number of lines of characters

is usually only about 25 and the positioning of the characters is restricted to a fixed matrix format. Vertical and horizontal lines cannot be drawn. Fairly extensive modification of the display formats in chapter 4 will be necessary if this type of device is used.

A fast typing or printing device offers the advantage of hard copy and the restriction on the number of lines is no longer present. However the use of paper each time a display is generated may preclude the free and unrestricted use of displays that a CRT device permits.

The choice of display device is therefore a tradeoff between cost and benefit, and must also be influenced by the other applications of the computer. Furthermore if the chosen display device does not have a keyboard, the cost of a teleprinter or a separate keyboard will also need to be taken into consideration.

Other items of necessary hardware include the following:-

a) A hard-copy device for printing out the timetable, if this function cannot be performed by the display device.

b) A data input device. For a central computer, a card or paper tape reader will be necessary for reading in the initial data. A school computer or terminal permits the direct entry of data on-line if this mode of entry is preferred.

c) A back-up device. A magnetic tape unit or paper tape punch is necessary for the production of back-up copies of the timetable data.

7.3.2 Program

Further development work on the current interactive timetabling program will consist mainly of making the system easier for a teacher to use. In particular, the following improvements could be made:-

a) Automation of the more complex requirement-alteration pro-

cedures, such as the use of dummy items, dummy requirements, and distribution groups.

b) Simplification of the processes of constraint relaxing and reintroduction, and of preparation for printout.

These and other enhancements will increase the size of the program. If a small computer is to be used, attention must be paid to the extra core required. It may be necessary to eliminate some of the lesser used features of the system to avoid exceeding the limited core space available.

The system will also need to be modified to suit the chosen hardware configuration. If a small computer is being used, the system will need to be reprogrammed in the assembly language of that machine. Otherwise reprogramming in a high-level language (with perhaps assembler subroutines for speed) would give the advantages of machine independence and ease of maintenance. A high-level language version would also be easier than an assembler version to translate into other assembly languages.

Apart from the language translation necessary, the display-generating program modules will have to be modified to produce display formats to suit the display device being used. Modifications to the printout programs will also be necessary for similar reasons.

7.3.3 Training

When the system is available to schools it will be necessary to train new users in all aspects of the use of the system. The training must include:-

a) Technical details of the operation of the system, such as the formats of the commands and displays.

b) Details of interactive timetabling procedure. This will include how and when to use the various operations available in the system to solve various problems that may arise.

c) Attention must also be paid to the process of preparing the block timetable before computer construction is begun. If

complex blocking structures can be avoided, less time will be needed for construction and the number of constraint relaxations necessary will be reduced. Avoidance of arbitrary changes to the requirements during construction should be encouraged as far as possible.

User manuals will be necessary and forms for data input will assist the new user in the preparation of the requirements for his school. Indeed forms will be necessary if data input is via cards. For the first few construction runs the new user will have to be guided through the entire timetabling procedure by an experienced person. Later he should be able to carry out most of the construction himself but he will need an advisor to assist him through the more difficult areas.

7.3.4 Maintenance

When the system is in operation, further changes will be necessary to the programs to eliminate bugs and to adapt the system to changing educational policies and ideas. The organisation which implemented the system will be in the best position to carry out this maintenance, provided that it receives the necessary information from the users. Maintenance can be complicated by the presence of a large number of different versions of the system, for different computers and with different facilities and features. If a common language is used, different versions can be generated from a master program in modular format. Maintenance is then carried out by modifying this master program, then generating the various versions required by extracting the appropriate modules. The use of a high-level language, as well as facilitating modification to the program, also permits the use of this 'modular' approach for generating different versions.

CHAPTER 8

CONCLUSION

This thesis has presented a new approach to constructing school timetables by computer. The significant features of this approach are:

a) It is interactive. The superiority of an interactive approach was argued in chapters 2 and 3. The basic arguments are
(1) The interactive approach is superior to the all-manual method, because it eliminates a large proportion of the tedium, frustration and error-proneness associated with the manual method.
(2) The interactive approach is superior to non-interactive computer methods because it allows 'human' decisions to be made when they should be made. Because of this it is able to produce better quality timetables.

b) Displays giving relevant information about the timetable form one essential feature of the approach. Such displays enable the timetabler to quickly observe the state of the whole timetable or of the section in which he is interested. Associated with the displays are operations which enable a wide variety of elementary changes to be made to the timetable.

c) The other essential feature is the provision of techniques to deal with difficult timetabling situations. Some situations which would otherwise cause difficulty can be detected and rectified at an early stage by the application of infeasibility testing and mutually clashing set searching. Others, which cannot be avoided, can be resolved by the use of the tree-search technique. The tree search forms the basis of an interactive method for solving even more difficult problems. It can also be used in conjunction with displays and other operations as a versatile means for rectifying undesirable features in the timetable.

Experience has shown that the success of the system and the willingness of the timetabler to use it depends on the feeling of progress that the timetabler must have at all times. The timetabler must not reach a situation in which much time is wasted and little useful construction is carried out, and he must not be forced into taking backward steps. To achieve the goal of ensuring smooth progress throughout, the displays and computer techniques available to the timetabler must be fast, effective, versatile and easy to use. Achieving these objectives was not a straight-forward process. This thesis has highlighted the problems that were encountered in the search for methods which work in practice. For example many display format designs which appeared promising were discarded. The development of display formats that could be used in interaction was hampered both by practical constraints such as screen size and by the poorly understood and complex nature of the human information processing involved in even simple tasks. Fully comprehensive methods for detecting infeasibilities were found to be impractical. And one of the simpler tree-searching algorithms was found to work best.

The success of the techniques presented is shown in the results of their application to Riccarton High School's timetabling problem. It is inevitable that timetabling for schools in general will be carried out by computers in the future. The trend in education today is to increase the choice of subjects offered to pupils to the extent that a pupil can choose virtually any combination of the subjects offered and can take any of these subjects at any year level. This trend will naturally make timetabling problems more difficult. On the other hand, computer costs are decreasing and computers are becoming more widely available. Use of computer timetabling systems will not only enable better timetables to be made more easily, but will also help to define how far the process of increasing the breadth of subject choice can go before the timetable is jeopardised. It is expected that the techniques described in this thesis will be accepted as a basis for future practical timetabling systems.

REFERENCES

AKKOYUNLU,E.A.(1973); "A linear algorithm for computing the optimum university timetable". Computer Journal 16, 4, pp 347-350.

ALMOND,Mary(1966); "An algorithm for constructing university timetables". Computer Journal 8, 4, pp 331-340.

ALMOND,Mary(1969); "A university faculty timetable". Computer Journal 12, pp 215-217.

ANDREAE,J.H., GALE,N.J., HENDERSON,K.D., and PLATTS,R.W.(1972); "Developing an interactive aid for school timetabling". Proceedings of the Third National Computer Conference vol 2, pp 182-202, N.Z. Computer Society.

APPLEBY,J.S., BLAKE,D.V., and NEWMAN,E.A.(1961); "Techniques for producing school timetables on a computer and their application to other scheduling problems". Computer Journal 3, 4, pp 237-245.

BARRACLOUGH,Elizabeth D.(1965); "The application of a digital computer to the construction of timetables". Computer Journal 8, 2, pp 136-146.

BERGE,C.(1962); "The Theory of Graphs and Its Applications". Methuen, London.

BERGHUIS,J., van der HEIDEN,A.J., and BAKKER,R.(1964); "The preparation of school timetables by electronic computer". BIT 4, 2, pp 106-114.

BRITTAN,J.N.G. and FARLEY,F.J.M.(1971); "College timetable construction by computer". Computer Journal 14, 4, pp 361-365.

BRODER,S.(1964); "Final examination scheduling". Communications of the ACM 7, 8, pp 494-498.

CHERNIAVSKY,A.L.(1972); "A program for timetable compilation by a look-ahead method". Artificial Intelligence 3, pp 61-76.

CLEARY,J.G.(1972); Internal memorandum. Department of Electrical Engineering, University of Canterbury, Christchurch, New Zealand.

COLE,A.J.(1964); "The preparation of examination time-tables using a small-store computer". Computer Journal 7, 2, pp 117-121.

CSIMA,J.(1965); "Investigations on a timetable problem". Ph.D. Thesis, Institute of Computer Science, University of Toronto.

CSIMA,J., and GOTLIEB,C.C.(1964); "Tests on a computer method for constructing school timetables". Communications of the ACM 7, 3, pp 160-163.

DAS,S.R.(1973); "On a new approach for finding all the modified cut-sets in an incompatibility graph". IEEE Transactions on Computers C22, 2, pp 187-193.

DEMPSTER,M.A.H.(1968); "On the Gotlieb-Csima time-tabling algorithm". Canadian Journal of Mathematics 20, pp 103-119.

DEMPSTER,M.A.H.(1971); "Two algorithms for the time-table problem", in Combinatorial Theory and its Applications, ed. D.J.A. Welsh, pp 63-85.

DEUTSCH,J.P.A.(1966); "A short cut for certain combinatorial problems". Proc. 1966 British Joint Computer Conference 19, pp 45-54.

DUNCAN,A.K.(1965); "Further results on the computer construction of school timetables". Communications of the ACM 8, 1, p 72.

ERNST,G.W., and NEWELL,A.(1969); "GPS: A case study in generality and problem solving". ACM Monograph, Academic Press, New York.

GELERNTER,H.(1959); "Realization of a geometry theorem proving machine", in Computers and Thought, eds. Feigenbaum and Feldman, McGraw-Hill, New York.

GELERNTER,H., HANSEN,J.R., and LOVELAND,D.W.(1960); "Empirical explorations of the geometry theorem proving machine", in Computers and Thought, eds. Feigenbaum and Feldman, McGraw-Hill, New York.

GOLOMB,S.W., and BAUMERT,L.D.(1965); "Backtrack programming". Journal of the Association for Computing Machinery 12, 4, pp 516-524.

GOTLIEB,C.C.(1963); "The construction of class-teacher time-tables". Proc. IFIP Congress, Amsterdam, pp 73-77.

HALL,P.(1935); "On representatives of subsets". Journal of the London Mathematical Society 10, 1, pp 26-30.

HARARY,F.(1969); "Graph Theory". Addison-Wesley.

HENDERSON,K.L.(1974); private communication.

HIGGENS,L.G., and ANDREAE,J.H.(1970); "A computer aid for the preparation of school timetables". Radio, Electronics and Communications (N.Z.) 24, 11, pp 13-16.

JOHNSTON,H.C., and WOLFENDEN,K.(1968); "Computer aided construction of school timetables". Proc. IFIP Congress (Edinburgh).

KONIG,D.(1950); "Theorie der endlichen und unendlichen graphen". Chelsea, New York.

KUHN,H.W.(1955); "The Hungarian method for the assignment problem". Nav. Res. Log. Quart. 2, pp 83-97.

LAWLER,E.L., and WOOD,D.E.(1966); "Branch-and-bound methods: a survey". Operations Research 14, 4, pp 699-719.

LAWRIE,N.L.(1966); "Linear programming as an aid to school timetabling". Proc. 1966 British Joint Computer Conference 19, pp 16-22.

LAWRIE,N.L.(1968); "School timetabling by computer", in Aspects of Educational Technology, vol II, Methuen, London.

LAWRIE,N.L.(1969); "An integer linear programming model of a school timetabling problem". Computer Journal 12, 4, pp 307-316.

LAZAK,D.(1969); "A heuristic approach to algorithms intended for the solution of the timetable problem". Computing 4, pp 359-377.

LEWIS,C.F.(1961); "The school timetable". Cambridge University Press.

LIONS,J.(1966a); "Matrix reduction using the Hungarian method for the generation of school timetables". Communications of the ACM 9, 5, pp 349-354.

LIONS,J.(1966b); "A counter-example for Gotlieb's method for the construction of school timetables". Communications of the ACM 9, 9, p 697.

LIONS,J.(1967); "The Ontario school scheduling program". Computer Journal 10, 1, pp 14-21.

LIONS,J.(1968); "A generalization of a method for the construction of class/teacher timetables". Proc. IFIP Congress (Edinburgh).

LIONS,J.(1971); "Some results concerning the reduction of binary matrices". Journal of the Association for Computing Machinery 18, 3, pp 424-430.

MILLER,G.A.(1967); "The magical number seven, plus or minus two", in "The Psychology of Communication", Basic Books, Pelican (1970).

MULLIGAN,G.D., and CORNEIL,D.G.(1972); "Corrections to Bierstone's algorithm for generating cliques". Journal of the Association for Computing Machinery 19, 2, pp 244-247.

NEISSER,U.(1964); "Visual search". Scientific American, June.

NEWELL,A., and SIMON,H.A.(1972); "Human Problem Solving". Prentice-Hall.

OLIVER,I.(1968); "Tree-searching school timetables". The Australian Computer Journal 1, 3, pp 153-157.

PLATTS,R.W.(1973); "A school timetabling program", in Man-Machine Studies Rept UC-DSE/2, ed. J.H. Andreae, Department of Electrical Engineering, University of Canterbury, New Zealand.

RYAN,D.M.(1969); "A survey of computer-aided timetable construction". Proc. Weekend Seminar: "The Role of the Computer in the Secondary School", Australian Comp. Soc. and Aust. Assoc. Math. Teachers, August 15-17 15-17, pp 158-165.

SAMUEL,A.L.(1969); "Some studies in machine learning using the game of checkers II Recent Progress". Annual Review of Automatic Programming 6, 1.

SLAGLE,J.R.(1971); "Artificial Intelligence — The heuristic programming approach". McGraw-Hill, New York.

SCOTT,R.A.(1969); "Timetabling", N.Z. Department of Education.

WELSH,D.J.A., and POWELL,M.B.(1967); "An upper bound for the chromatic number of a graph and its application to timetabling problems". Computer Journal 10, pp 85-86.

de WERRA,D.(1971); "Construction of school timetables by flow methods". Canadian Journal of Operational Research and Information Processing 9, 1, pp 12-22.

WOOD,D.C.(1968); "A system for computing university examination timetables". Computer Journal 11, pp 41-47.

WOOD,D.C.(1969); "A technique for colouring a graph applicable to large scale timetabling problems". Computer Journal 12, 4, pp 317-319.

YULE,A.P.(1968); "Extensions to the heuristic algorithm for University timetables". Computer Journal 10, pp 360-364.

APPENDIX A

SET DEFINING

The set defining system is used by certain commands in the timetabling system (see appendix C). The parameters for such commands include a string of characters which is interpreted by the set-defining subroutine, which in turn flags requirements, items or periods in the set so defined. This appendix defines the syntax of permissible strings in Backus-Naur form and describes their effects.

The string entered as a parameter to a command will be denoted by <set of requirements>, <set of items>, or <set of periods> depending on the type of set being defined.

<set of requirements> ::= <elementary set> |
<set of requirements>,<elementary set> |
<set of requirements>Ø<elementary set>

A set of requirements may consist of unions and/or intersections of elementary sets. Union is denoted by ',' and intersection by blank (Ø). Evaluation takes place strictly from left to right — there are no priorities.

<elementary set> ::= <basic set> | \<basic set>

The complement of a basic set may be formed by preceding its definition with a backslash (\). The complement consists of all requirements other than those in <basic set>.

<basic set> ::=

<requirement>
A requirement code defines a set consisting of the one requirement.

| <requirement>-<requirement>
Two requirement codes separated by '-' define the set consisting of all requirements in the list between and including the two specified.

| <teacher>

A teacher mnemonic defines the set of requirements involving the teacher.

| <teacher>-<teacher>

A range of teachers defines the set of requirements involving at least one of the teachers within the range.

| <class> | <class>-<class>

Similarly for classes.

| <classroom> | <classroom>-<classroom>

Similarly for classrooms.

| <subject> | <subject>-<subject>

Defines the set of requirements in which at least one of the specified subjects is taught.

| <day/prd> | <day/prd>-<day/prd>

(<day/prd> ::= <digit>/<digit>) Two digits separated by '/' is a <day/prd>. For example 3/2 represents day 3, period 2. A <day/prd> or a range of <day/prd>s defines the set of requirements that have at least one assignment present in any of the specified periods in the timetable.

| <teacher><subject>

Defines the set of requirements in which <teacher> takes <subject>.

| <numerical attribute>=<number>
| <numerical attribute>><number>
| <numerical attribute><<number>

Defines the set of requirements for which the specified numerical relationship is satisfied. <number> ::= 0 | 1 | 2 | 3 | ...

<numerical attribute> ::=

TS	total number of single-period assignments
TD	total number of double-period assignments
DN	distribution constraint reference
AP	allowed-period pattern reference
NS	number of single-period lessons remaining
ND	number of double-period lessons remaining
NO	total number of periods remaining to be allocated
PN	the value P-N
NP	the value N-P
XS	the number of single-period lessons in the timetable
XD	the number of double-period lessons in the timetable

Examples:

#I	The set consisting of the single requirement #I.
@A,@B,@C	The set {@A,@B,@C}.
@1-@6	All requirements between @1 and @6 in the list.
3A	All requirements involving class 3A.
BEART	All requirements in which BE teaches ART.
ART∅TD>0	All requirements containing double periods of ART.
2/1-2/7	All requirements which have assignments in Tuesday of the timetable.
\TD	All requirements that do not contain any TD rooms.

<set of items> The delimiters ',' '∅' and '\' may be used for union, intersection and complement as in requirement set definitions. The syntax is similar.

<basic set> ::= (for items)

<requirement>

Defines the set of items involved in <requirement>.

| <requirement>-<requirement>
Defines the set of items involved in any one of the requirements within the range.

| <teacher>
Defines the set consisting of the single element <teacher>.

| <teacher>-<teacher>
Defines the set consisting of all teachers in the range.

| <class> | <class>-<class>
| <classroom> | <classroom>-<classroom>
Similarly for classes and classrooms.

| <day/prd> | <day/prd>-<day/prd>
Defines the set of items occupied in any of the specified periods.

<set of periods> The delimiters ',' 'Ø' and '\' may be used for defining unions, intersections and complements of sets of periods.

<basic set> ::= (for periods)

<requirement> | <requirement>-<requirement>
Defines the set of periods in which at least one of the requirements has an assignment.

| <teacher> | <teacher>-<teacher>
| <class> | <class>-<class>
| <classroom> | <classroom>-<classroom>
Defines the set of periods in which at least one of the specified items is occupied.

| <teacher><subject>
Defines the set of periods in which <teacher> takes <subject>.

APPENDIX B

TIMETABLE DATA ORGANISATION

The structure of the timetable data stored in the computer is illustrated in figure B.1. The components of this data structure are described below.

a) Mnemonics. Each requirement, class, teacher and classroom type is allocated a two-character mnemonic. Subjects have three-character mnemonics. Lists of these mnemonics are maintained by the operations AR (alter requirements), AT (alter teachers), AG (alter groups), AC (alter classrooms), and AS (alter subjects).

b) Requirements. Each requirement consists of:-
(1) A list of classes.
(2) A list of teachers with associated subjects.
(3) A list of classroom types. With each type, the number of rooms (lives) of that type needed for the requirement is also included.
(4) The total number of single-period lessons and the total number of double-period lessons.
(5) The number of single-period lessons and the number of double-period lessons remaining to be entered into the timetable. One of these quantities is decreased by 1 each time an entry is made into the timetable.
(6) A numeric reference to an allowed-period pattern.
(7) A numeric reference to a distribution constraint.
(8) References to the distribution groups which involve the requirement.

c) Timetable. The timetable is stored as a two-dimensional class-period array. The element of the array for class c, period p contains a numeric reference to the requirement in which c is involved in period p. Each timetable entry also has a fix flag which, when set to 1, indicates to the program that the assignment cannot be unloaded or moved.

d) Allowed-Period Patterns. Each requirement references an entry in the list of allowed-period patterns. An allowed-period pattern is a row of binary digits, one digit for

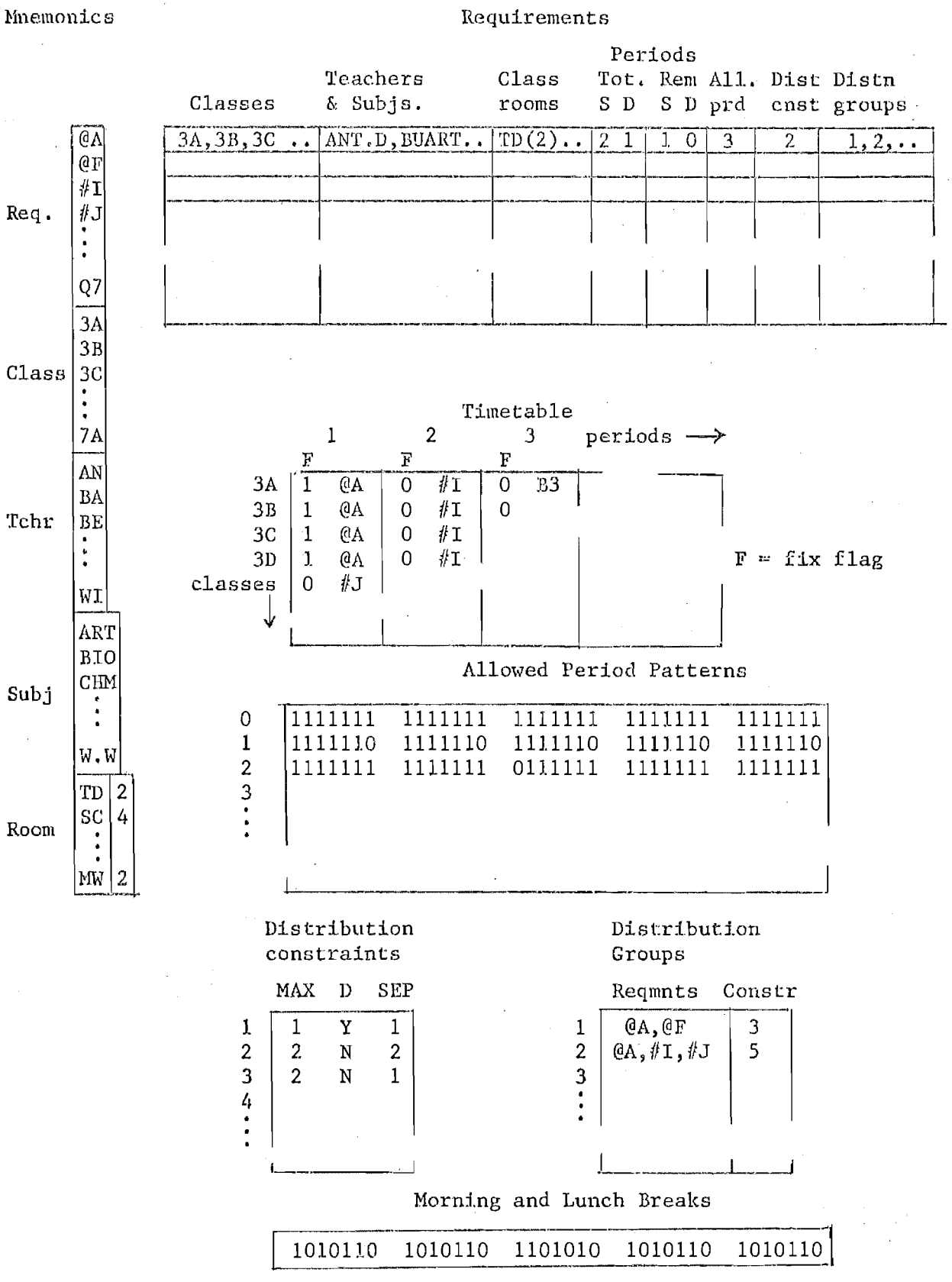


Figure B.1 Timetable Data Organisation

each period. A zero digit indicates that the corresponding period is not allowed, a '1' indicates that it is allowed. A requirement that references a particular pattern can only be loaded into periods for which the pattern digits are '1'.

e) Distribution constraints. A distribution constraint may be referenced either by a requirement or by a distribution group. Each constraint comprises three parameters:-
(1) MAX. The maximum permitted number of days which may contain two lessons of the requirement or distribution group.
(2) D. A flag indicating whether or not doubles are permitted.
(3) SEP. The minimum permitted separation between two lessons on the same day. Separation is measured in periods, with a morning or lunch break counting as one period.

f) Distribution Groups. Each distribution group contains a set of requirements together with a reference to a distribution constraint. The constraint applies collectively to all members of the group.

g) The total number of classrooms of each type in the school is stored to enable classroom clashing to be determined.

h) The positions of morning and lunch breaks are stored in a format similar to that of an allowed-period pattern. A zero in a period position indicates that a break follows that period.

APPENDIX C

TIMETABLING SYSTEM COMMANDS

C.1 General Housekeeping

DG Display class mnemonics (figure C.1). Also given in the display are the total number of lesson-periods for each class, and the total number of double-period assignments that each class is involved in.

DT Display teacher mnemonics (figure C.2). The number of teaching periods and the number of double periods are also included for each teacher.

DC Display multiple-life classroom mnemonics, together with the total number of lives for each classroom type (figure C.3).

DS Display subject mnemonics (figure C.4).

DP Display the list of allowed-period patterns referenced by the requirements (figure C.5). The top line, labelled 'BRK', shows the positions of morning and lunch breaks.

DD Display the list of distribution constraints (figure C.6).

AG Alter class mnemonics. Parameters given via the keyboard indicate the position in the list of mnemonics and the new mnemonic.

AT Alter teacher mnemonics.

AC Alter classroom mnemonics and associated life numbers.

AS Alter subject mnemonics.

AD Alter distribution constraint parameters.

C.2 Requirements

LR List all requirements (figure C.7). Each line in this display consists of, from left to right,

a) The requirement mnemonic or code.

b) The number of single- and double-period lessons. This

DG

NO	MN	PRDS	
1	3A	35	1
2	3B	35	1
3	3C	35	2
4	3D	35	2
5	3E	35	2
6	3F	35	2
7	3G	35	2
8	3H	35	1
9	3I	35	1
10	3J	35	1
11	3K	35	1
12	3S	35	1
13		0	0
14	4S	35	1
15	4A	35	2
16	4B	35	2
17	4C	35	2
18	4D	35	2
	[illegible]	35	2

Figure C.1

Classes

DT

NO	MN	PRDS	
1	HM	7	2
2	DP	6	2
3	HZ	12	1
4		0	0
5	BE	32	11
6	BK	32	3
7	BT	32	0
8	BR	30	5
9	BN	29	0
10	BU	25	1
11	CA	29	1
12	CL	32	1
13	CO	32	2
14	CR	22	1
15	CS	32	2
16	DA	25	1
17	DN	29	2
18	EA	32	1
	ED	31	0
		[illegible]	2

Figure C.2

Teachers

DC

NO	MN	LIFE NO
1	TD	2
2		0
3		0
4		0
5		0
6		0
7		0

Figure C.3

Classrooms

DS

NO	MN
1	ACC
2	ADM
3	ART
4	BIO
5	CHM
6	CLO
7	CPR
8	CPS
9	CST
10	ECN
11	ENG
12	FR.
13	GEO
14	GER
15	GST
16	HEC
17	HIS
	TDS

Figure C.4 Subjects

DP

BRK	1101010	1010110	1101010	1010110	1010110
0	1111111	1111111	1111111	1111111	1111111
1	1111101	1111111	1111111	1111111	1111111
2	1101110	1011111	1101111	1011111	1011110
3	1101100	1011111	1101111	1011111	1011110
4	0111110	0011111	0111111	0011111	0011111
5	0111111	1111111	1111111	1111111	1111111
6	1111011	1110111	1111011	1110111	1110111
7	0111111	1111111	1111111	1111111	1111110
8	1111111	1111111	1111111	1111111	1111110
9	0001110	0011110	0001111	0011110	0011110
10	0111110	0011111	0111111	0111111	0011111
11	0001110	0011110	0111111	0011110	0011110
12	0001110	0011110	0001111	0011110	0011111
13	0011110	0011110	0001111	[illegible]	
14	0001110	0011111	[illegible]		

Figure C.5 Allowed-Period Patterns

DD

NO	MAX	D	SEP
1	1	Y	3
2	2	Y	3
3	1	Y	0
4	2	Y	0
5	1	N	3
6	2	N	3
7			
8			

Figure C.6

Distribution Constraints

```
+A 3S  2D    6A,6B  HZENG COENG EAENG ELENG JNENG 21 0
+E 1S  1D    6A,6B  DPHIS BKPHX BRPHX EGHIS LYP.E LOART 2 0 +F 0
+F 1S  1D    6A,6B  DPHIS BEART BKPHX BRPHX EGHIS LYP.E 2 0 +E 0
+J 3S  1D    5D-6B 5M 5N  BUFR. CSCHM DNS.S HDMTH MAMTH PRMTH RSCHM FGACC
 6 0 +I 0 [3 2
+I 1S    6A,6B  BUFR. CSCHM DNS.S MAMTH PRMTH RSCHM FGACC 4 0 +J 0
+N 4S  1D    6A,6B  DNGEO EDMTH GNSEC HDMTH TNGEO 6 0
+T 4S  1D    6A  CLBIO DABIO HSBIO INGER SHT.D SDBIO 1TD 0 0
+U 4S    6B  PRCPS 0 0
+X 2S    6A,6B  GLMUS HNW.W LOART MOM.W WAP.E MW WW 7 0 +Y 4
+Y 1D    6A,6B  GNCST HNW.W LOART MOM.W WAP.E MW WW 8 0 +X 4
%A 2S    5A-5S 5M 5N 5O  BEART BTP.E EDMTH FLHEC HNW.W LYP.E LOART PRMTH
 SHM.W SPCLO THMTH MW WW 0 0 %C %D 1 %B %D 4 %B %C 4
%B 2S    5A-5S 5M 5N 5O  BEART BTP.E FLHEC GNTYP HNW.W LOART MNP.E SHM.W
 SPCLO WAP.E WIGEO WNTYP TP MW WW 0 0 %C %D 1 %A %D 4 %A %C 4
%C 2S    5A-5S 5M 5N 5O  BEART BTP.E EDMTH GNTYP HNW.W LOART PRMTH THMTH
 WIGEO WNTYP MLP.E TP 2S 0 %B %D 1 %A %D 1 %A %B 4
%D 2S    5A-5S 5M 5N 5O  BTP.E EDMTH FLHEC GNTYP PRMTH SHM.W SPCLO THMTH
 WAP.E WIGEO WNTYP MLP.E TP 25 0 %B %C 1 %A %C 1 %A %B 4
&T  5A-5J 5O  EDMTH GNTYP PRMTH THMTH WIGEO WNTYP TP 0 1 &U 1 &V 1
&U  5A-5J 5O  BTP.E EDMTH LYP.E PRMTH THMTH 0 0 &T 1
&V  5A-5J 5O  BTP.E GNTYP MNP.E WAP.E WIGEO WNTYP TP 1S 0 &T 1
&W  5G,5H 5M  BTP.E MLP.E 25 0
%X  5G,5H 5M  FLHEC SHM.W SPCLO 0 0 &X 0
&X  5G,5H 5M  FLHEC SHM.W SPCLO MW 0 0 %X 0
%Y  5I-5S 5N  BEART HNW.W LOART 0 0 &Y 0
&Y  5I-5S 5N  BEART HNW.W LOART WW 0 0 %Y 0
&Z  5I-5S 5N  BTP.E WAP.E MLP.E 25 0
@N 2S  1D    4A-4D  HZIDS HNW.W MOM.W SPHEC FGCST MW WW 3 0
@R 2S  1D    4E-4G  FLHEC ANW.W SHM.W SPCLO MW WW 0 0
@U 2S  1D    4H-4J  FLHEC MOM.W ANW.W SPCLO MW WW 0 0
@L 1S    3S-4S  MOM.W ANW.W 0 0 @M 0
@M 3S    3S-4S  MOM.W ANW.W MW WW 0 0 @L 0
@C 2S  1D    3C-3E  FLHEC HNW.W MOM.W SPCLO MW WW 0 4
@F 2S  1D    3F-3H  FLHEC HNW.W SHM.W SPCLO MW WW 0 4
@I 2S  1D    3I-3K  FLHEC HNW.W MOM.W SPCLO MW WW 0 4
&N 5S  1D    5C-5J 5M 5N 5O  BKPHX CAGER CSSCI DNGEO EAFR. NSSCI SKCPR TNGEO
 FGCPR 0 2
&E 4S  1D    5A-5J 5O  BNHIS COHIS EGHIS ANT.D RYGEO SHT.D LTGEO 2TD 13 0
&A 1S    5H-5J 5O  ELENG LEENG TYENG 22 0 &B 2
&B 6S    5H-5J 5O  ELENG JNENG LEENG 22 2 &A 2
&I 5S  1D    5H-5J  CRMTH MAMTH MNMTH 17 2
&J 5S    5S 5O  TBMTH 0 1
$N 4S    4A-4D  BUFR. EAFR. INGER SKCST 0 0
$R 4S    4E-4G  GNTYP ANT.D SKCST TP 1TD 0 0
$U 4S    4H-4J  GNCST ANT.D WNTYP MMMST TP 1TD 0 0
M@ 1D    3S-4S  FLHEC SPHEC 0 0 N@ 0
N@ 2S    3S-4S  FLHEC GLMUS 7 0 M@ 0
$H 4S    3S-4S  GNTYP FGCST 0 0
$C 4S    3C-3E  EAFR. ANT.D SKCST WNTYP TP 1TD 0 0
$F 4S    3F-3H  BUFR. MOT.D SKTYP FGCST TP 1TD 0 0
$I 4S    3I-3K  SHT.D SKTYP FGCST TP 1TD 0 0
@@ 1S    0X  HZ CA HS MN RY TY TB WS FG 0 0
$A 4S  1D    7A,7B 7P 7Q  JNENG LEENG 0 0
$E 2S  2D    7A,7B 7P 7Q  COHIS RSCHM 0 0
$I 3S    7B 7Q  BUFR. 0 0
$J 2S  2D    7A 7P  BRPHX PRMTH WNACC 0 0
$O 3S    7B 7P  INGER 0 0
$M 3S    7A 7Q  CLBIO SKECN TBADM 0 0 $N 3
$N 3S    7A 7Q  CLBIO TBADM 0 0 $M 3
$T 4S  1D    7A,7B 7P 7Q  MAMTH WIGEO 0 0
$W 2S    7A,7B 7P 7Q  CLGST 0 0 $X $Y 0
$X 1D    7A,7B 7P 7Q  BRGST 0 0 $W $Y 0
$Y 1S    7A,7B 7P 7Q  EGGST 0 0 $W $X 0
N9 3S    4A,4B  BTP.E LYP.E 0 0
P9 3S    4C,4D  BTP.E WAP.E 0 0
R9 3S    4E,4F  MNP.E WAP.E 19 0
T9 3S    4G,4H  LYP.E MLP.E 25 0
V9 3S    4I,4J  BTP.E WAP.E 0 0
M9 3S    4S  LYP.E 0 0
A9 3S    3A,3B  BTP.E WAP.E 0 0
C9 3S    3C,3D  BTP.E LYP.E 0 0
```

Figure C.7 Requirement List Display

is given in the form $n_1 X n_2$ where
n_1 = the total number of lessons
X = S for single-period
D for double-period
n_2 = the number of lessons remaining to be entered into the timetable. Omitted if zero.

c) The classes involved in the requirement. Classes may be expressed as a range, e.g. 5A-5F means 5A, 5B, 5C, 5D, 5E, 5F.

d) The teachers involved in the requirement, together with their associated subjects.

e) Classrooms involved in the requirement. Multiple-life classroom-types are preceded by the number of lives required.

f) The numeric reference to an allowed-period pattern.

g) The numeric reference to the distribution constraint which applies to the requirement alone.

h) A list of other members of distribution groups which include the displayed requirement. The displayed requirement itself is not shown in any of the groups in this list. Each group is followed by the reference to the distribution constraint applying to that group.

SR List all requirements in a given set. The set is defined by the set-defining system (appendix A) from the parameter string that is requested by this command.

IN Insert a given requirement into the requirement list in a position immediately preceding another specified requirement. This command enables the ordering of the requirement list to be altered.

AR Alter existing requirements or create new requirements. Each alteration sub-command must be in one of the following formats:-

<set of requirements>:<change>,<change>,...,<change>
Applies the specified changes to all members of the specified set of requirements. Changes are defined below.

or <set of requirements>:

Deletes all members of the specified set of requirements.

or <new requirement>:<change>,<change>,...,<change>

Creates a new requirement. A <new requirement> is any valid requirement code not already in the mnemonic list.

or <set of requirements>:<distn group>,<distn group>,..., <distn group>

Creates one or more distribution groups.

<change> ::=

<teacher>

Adds the teacher to each requirement in the set defined. The associated subject becomes that of the teacher most recently deleted.

| \<teacher>

Deletes the teacher from each requirement.

| <teacher><subject>

Adds the teacher with the associated subject to each requirement.

| <class>

Adds the class to each requirement.

| <class>-<class>

Adds all classes in the range to each requirement.

| \<class>

Deletes the class from each requirement.

| \<class>-<class>

Deletes all classes in the range from each requirement.

| <classroom>

Increases by one the required number of lives of the classroom in each requirement.

| \<classroom>

Decreases by one the required number of lives

of the classroom in each requirement.

| <subject>

Assigns the subject to all teachers without associated subjects in each requirement.

| <day/prd>

Allows the period for each requirement.

| <day/prd>-<day/prd>

Allows all periods in the range for each requirement.

| <day/prd> | <day/prd>-<day/prd>

Disallows the period or range of periods for each requirement.

| TS=<number>

Sets the total number of single-period assignments.

| TD=<number>

Sets the total number of double-period assignments.

| DN=<number>

Sets the distribution constraint reference for each requirement to <number>. Distribution constraints themselves are maintained by use of the command AD.

| (<requirement>,<requirement>,...,<requirement>)

Adds the class, teachers with subjects, classrooms and disallowed periods of each requirement in the bracketed list to each requirement in the defined set.

<distn group> ::= <requirement>,<requirement>,...,
<requirement>,<number>

Creates a distribution group which contains all the requirements listed and which is controlled by the distribution constraint referenced by <number>.

C.3 Timetable Displays

CR A string defining a set of requirements is given as a parameter. Distribution, free periods, disallowed periods and clashes are displayed for each requirement in the set (figure C.8). Symbols in the display are interpreted as follows:

□ or [ ] free period.

— or -- disallowed period.

\- a clashing assignment is fixed in that period.

✳ or ╳ the requirement displayed has an assignment in that period.

o an assignment in that period is in the same distribution group as the displayed requirement.

DR Similar to CR, except that clashes are not displayed (figure C.9).

DI The parameter string following this command defines a set of items. The assignments in the timetable for each item in the set are displayed (figure C.10).

GX Display class-period timetable (figure 4.4, chapter 4).

TX Display teacher-period timetable (figure 4.5, chapter 4).

CX Display classroom-period timetable. Format is similar to that of teacher-period timetable.

D∅ (∅ represents blank) If a requirement is specified as the parameter, this command displays distribution, free-period, disallowed-period and clash information for the requirement (figure 4.8, chapter 4). Clashes are formatted alongside the items for the requirement.
If an item is specified as the argument, this command displays

a) the assignments in the timetable for the item,
b) the distribution, free periods and disallowed periods for requirements involving that item. (Figure C.11).

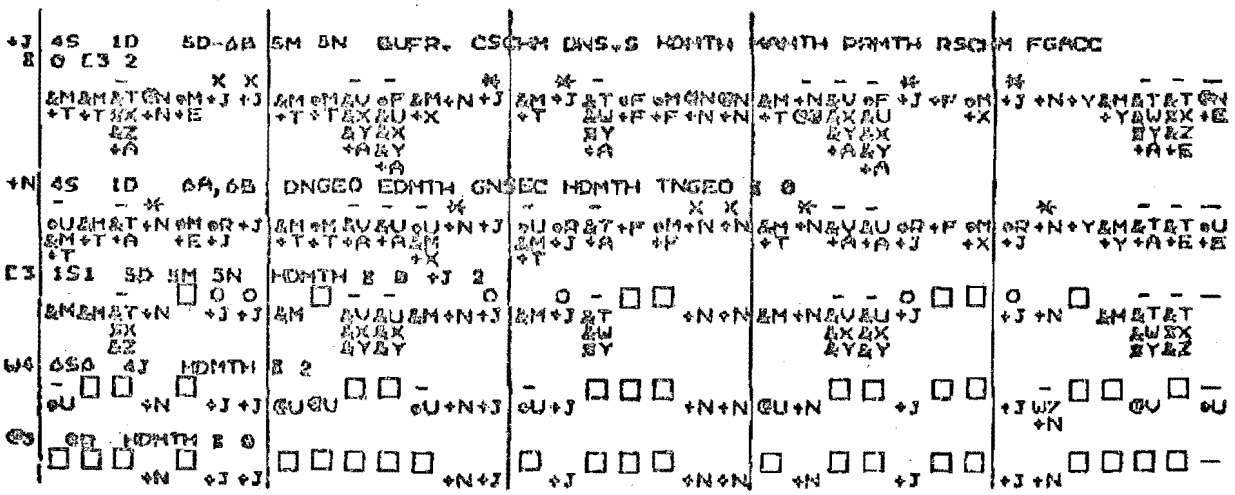


Figure C.8 'CR' Display

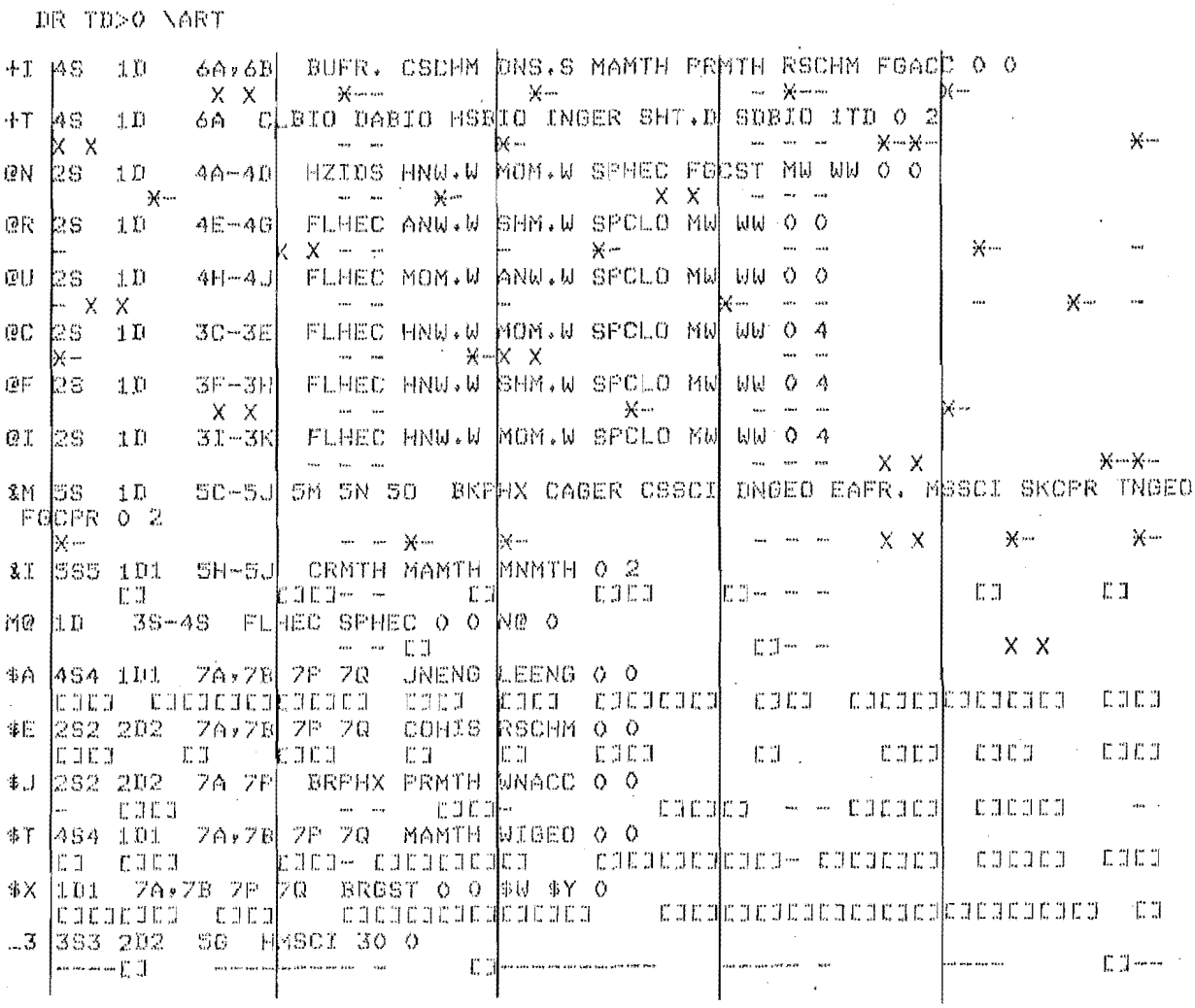


Figure C.9 'DR' Display

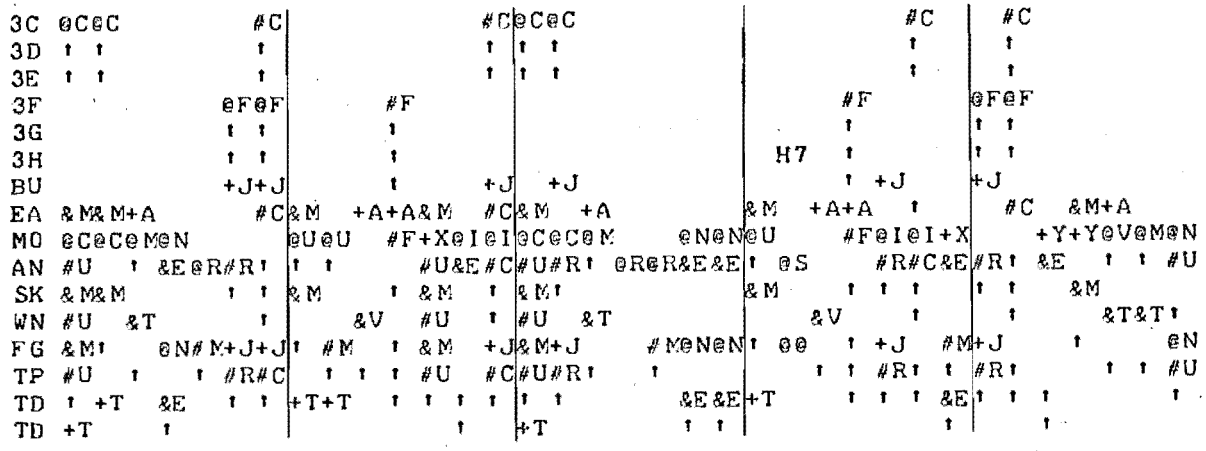


Figure C.10 'DI' Display for Items in #C and #F

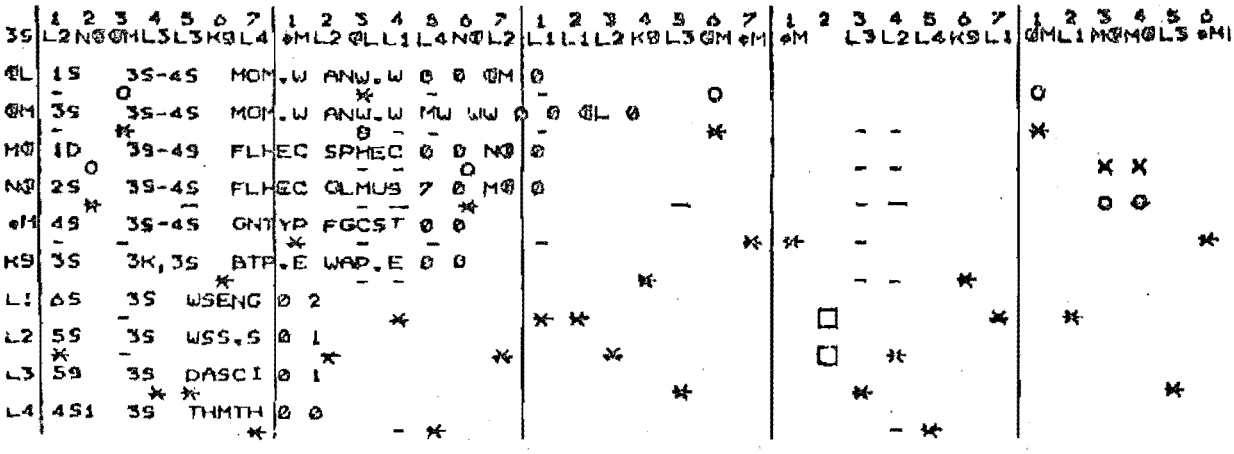


Figure C.11 'DØ' Display for Class 3S

C$\emptyset$ An item is specified as the parameter. This command produces a display similar in format to that of D$\emptyset$ (item) except that clashes are given for every requirement displayed. (Figure 4.9, chapter 4).

C.4 Basic Timetable Operations

L$\emptyset$ Load a specified requirement into a specified period. Clashes will be displaced if this is permitted by the user (see section 4.4.1).

U$\emptyset$ Unload a specified assignment from a specified period.

F$\emptyset$ Fix an assignment.

X$\emptyset$ Unfix an assignment.

C.5 Advanced Timetable Operations

LX Two strings must be provided as parameters, one defining a set of requirements, the other a set of periods. This command attempts to load each member of the set of requirements into each period defined in the set of periods. No distribution check is undertaken.

LD Operates in the same way as LX, except that distribution is checked as the loading takes place. Assignments will be loaded into only those periods which satisfy all distribution constraints.

UX Parameter strings define a set of requirements and a set of periods. This command unloads all assignments belonging to each member of the requirement set from each period in the period set.

SP Two lists of periods are given as parameters. This command moves all assignments from each period in the first list to the corresponding period in the second list. This enables timetable periods to be interchanged or rotated.

TS Tree search. Parameters that must be given are the requirement to be entered into the timetable, and the maximum depth of the search. If a solution is found, the

stack describing the solution interchange is printed out (figure C.12). If no solution is found, the maximum depth is increased by three and the search is continued.

```
←
C:18:37
  4 #F 1/4 &E,@N,
  7 @N 4/1 #I,+A,
  9 +A 1/5 +E,
 10 +E 4/1
 13 #I 5/2 #F,@S,
 15 @S 5/5 @U,
 16 @U 5/2
 18 #F 2/4 #R,
 19 #R 4/1
 22 &E 2/2 @R,+F,
 24 +F 3/4 +E,
 27 +E 4/1 +N,+E,
 28 +E 2/1
 29 +N 2/2
 33 @R 1/4 @M,@R,%D,
 35 %D 1/3 %C,
 36 %C 1/5
 37 @R 2/1
 38 @M 1/3
```

Figure C.12 Tree Search Solution Print-out

PS Print tree-search stack. This command is used during the operation of the tree search to assess progress.

FL Print flagged requirements (see section 6.14.2).

AB Abort tree search.

CC Determine whether the value of P-N for any requirement is less than zero, or whether any item is infeasible (see section 5.2.10).

MC Search for and list all maximal mutually clashing sets of requirements whose total lesson-periods exceed a specified number (see section 5.3).

HT Automatic timetable construction. A heuristic procedure is used to select requirements and to allocate assignments to periods. Selection is performed on the basis of minimum P-N, or on maximum clash number if all P-N's are high. The clash number of a requirement r_o is defined as

$$CLASH(r_o) = \sum_{r \neq r_o} N_r C_{r_o r}$$

where N_r is the number of lesson-periods required by requirement r

$C_{r_o r}$ is an element of the clash matrix (definition (definition 2.5).

Lessons are allocated to free periods in which 'interference' is minimum. Interference is caused if the allocation of a lesson of r_o to period p results in that period being no longer free for another requirement r'. Interference values are weighted and summed to give the total interference value for r_o in period p. The weighting factor is designed to greatly increase the interference contributions of requirements r' with low P-N's.
If free periods cannot be found for the selected requirement, tree searching is performed up to a limited depth. If this fails, automatic timetabling halts and control is returned to the user.

CT Operates in the same manner as HT, except that the infeasibility testing procedure (section 5.2.10) is used as a further alternative means for selecting requirements when all P-N's are high. If an infeasible set of requirements is found, then each member of the set is allocated in turn as though it had been selected individually.

C.6 Miscellaneous Commands

MN Terminate execution of the timetabling system and return control to the system monitor.

PT Punch a paper-tape back-up copy of the timetable data in core or read in a paper-tape back-up.

ØØ Allow user comments to be typed on the teletype or the display.

RT Read all future commands and input from paper tape. This command enables data prepared off-line to be read in.

KB Used in paper tape input to return control to the keyboard.

APPENDIX D

CLASSROOM ASSIGNING AND PRINTOUT

A separate program overlay assigns classroom numbers to a finished timetable and prints the timetable out in easily legible form. Extra data is needed for this program and consists of:-

a) Full names of teachers.

b) Full names of classes.

c) Classroom numbers.

d) Full names of subjects.

This data is maintained by 'alter' and 'display' commands similar to those in the main system.

Classroom assignments are stored in a two-dimensional teacher-period array. The classroom-assigning command accepts three parameters from the keyboard:-

a) a classroom number

b) a set of periods

c) a teacher.

The room is assigned to the teacher in each of the periods in the specified set. The use of the set defining system enables classrooms to be assigned on the basis of subject taught, day, requirement, or any combination of these. Classrooms may also be deleted from periods. No classroom may be assigned to more than one teacher in a period.

Commands are available to produce displays showing:-

a) Class and subject taken by a given teacher in each period, together with the currently free classrooms for each period (figure D.1). This display assists the timetabler in choosing a suitable free room for the teacher and the subject.

b) Teacher, class and subject to which a given room has been allocated in each period of the week (figure D.2). This display enables the timetabler to check that the room has

been allocated correctly.

The timetable can be printed out in four formats (figures D.3 to D.6). In addition, a condensed version of the class master timetable (figure D.7) is available for use off-line by heads of departments in assigning science laboratories and other specialist rooms.

HENDERSN

	1	2	3	4	5	6	7
MON		6HD MATHS E7 B5 C5 C7 E6 F4 H5	R/O MATHS D5 E7 F3 F6 H4		4WA MATHS E7 C4 D3 F7 H7 P1	5IN MATHS E7 E5 F3 F5 G3 P5	5IN MATHS E7 D3 E1 F1 G1
	B7			D4 D5 H5 M2			
TUE	R/O MATHS C1 C3 C5 C6 C7 D4 E6 H5	6HD MATHS E7 B7 D5 H5	B5 E7 P1 P2	D5 F5 F7 H5 M1		5IN MATHS E7 C7 D3 F1 P1	4WA MATHS E7 D5 F3 G7
WED	R/O MATHS B5 E7	5IN MATHS E7 B7 D3	B6 B7 D4 D5 E6 M1	6HD MATHS E7 C3 C5 C7 D4 H6 H7	6HD MATHS E7 C5 C7 H4	4WA MATHS E7 F7 H7 M1	R/O MATHS D4 D5 G6 H5 P3
THU	4WA MATHS E7 C3 C5 C6 C7 D4 E3 E4 F3	6HD MATHS E7 C3 D3 H7	D3 D5 F5 H1	4WA MATHS E7 D5 G1 G4 G6 M1 P1	R/O MATHS C5 D5 E7	C5 D5	5IN MATHS E7 B7 D3 E1 G1 H4
FRI	6HD MATHS E7 B5 C6 C7 D5 H3 H5 P5	5IN MATHS E7 C5 F5 P1 P3	C5 D3 D4 E5 F6 P1	R/O MATHS E6 E7 F1 G1 H5 H7 M1	4WA MATHS E7 C3 D3 P1	5IN MATHS E7 D3 E3 H6	B6

Figure D.1 Free-Classroom Display for Given Teacher

MON	DUNCAN 5RY-5CA GEOG	DUNCAN 6DN GEOG	DUNCAN 3BE S.S.	DUNCAN 4BT S.S.	DUNCAN 3BE S.S.	DUNCAN F.6 S.S.	DUNCAN F.6 S.S.
TUE	LIEUTHTE 5HS-5CA GEOG	DUNCAN 6DN GEOG	DUNCAN F.6 S.S.	LIEUTHTE 4WA S.S.	DUNCAN 5RY-5CA GEOG	DUNCAN 3BE S.S.	DUNCAN 4BT S.S.
WED	DUNCAN 5RY-5CA GEOG	DUNCAN F.6 S.S.	LIEUTHTE 3MCS S.S.	DUNCAN 6DN GEOG	DUNCAN 6DN GEOG	DUNCAN 4BT S.S.	DUNCAN 3BE S.S.
THU	LIEUTHTE 5HS-5CA GEOG	DUNCAN 6DN GEOG	LIEUTHTE 4WA S.S.		DUNCAN 5RY-5CA GEOG	DUNCAN 5RY-5CA GEOG	DUNCAN F.6 S.S.
FRI	DUNCAN 6DN GEOG	DUNCAN F.6 S.S.	DUNCAN 5RY-5CA GEOG	DUNCAN 3BE S.S.	DUNCAN 4BT S.S.	DUNCAN 4BT S.S.	DUNCAN 5RY-5CA GEOG

Figure D.2 Teacher/Class/Subject Display for Room

Figure D.3 Class Master Timetable

MONDAY

	1	2	3	4	5	6	7
3MCN	MCNEILE MATHS E1	CAMPBELL GER F3	MGOLDRCK ENG P1	BROWN-RC S.S. H3	HARKNESS SCI H5	MCNEILE MATHS E1	BOOTH P.E.
3EA	BROWN-MD SCI F4	PORTER MATHS E3	CONRADSN ENG G3	THURSTON S.S. G5	WILLIAMS ENG P3	INNES GER F1	WALLS P.E.
3CS	BURGESS ENG H7	BLACK MATHS F5	GALE MUSIC M1	BLACK MATHS F5	CROSS SCI C1	WILKNSON S.S. G7	EAGLE FR. P2
3EG	EGGLESTN S.S. G1	BENNETT ART B6	BENNETT ART B6	GALE MUSIC M1	CAMPBELL ENG F3	EGGLESTN S.S. G1	ANDERSON T.D. D5
3BE	EDIE MATHS E5	MCNEILE P.E. E1 WALLS P.E.	DUNCAN S.S. G4	EDIE MATHS E5	DUNCAN S.S. G4	WINN ENG P2	SKINNER C.ST. C4 WINN TYP C5
3MCS	RYAN ENG H1	" " " "	LIEUTHTE S.S. H1	BURGESS FR. H7	MCSWENEY SCI D4	FLEMING H.EC. C7	FLEMING H.EC. C7
3TY	BOOTH P.E.	MCSWENEY SCI D4	BROWN-RC S.S. H3	MOORE T.D. D3	BROWN-RC S.S. H3	HEMINGSN W.W. D6	HEMINGSN W.W. D6
3WI	WALLS P.E.	INNES ENG F1	MCAULIFF MATHS F7	SKINNER TYP C5 FINNEGAN C.ST. C4	SIDAWAY SCI D1	SHERRARD M.W. D7 SIEPKES CLO C6	SHERRARD M.W. D7 SIEPKES CLO C6
3RS	FLEMING H.EC. C7	FAASS S.S. G6	RICHARDS SCI C3	THOMAS MATHS E6	LEAN ENG M1	BOOTH P.E.	GALE MUSIC M1
3SK	HEMINGSN W.W. D6	SIDAWAY SCI D1	SIDAWAY SCI D1	BENNETT ART B6	THURSTON S.S. G5	WALLS P.E.	EDIE MATHS E5
3CR	MOORE M.W. D7 SIEPKES CLO C6	CLARK SCI E4	CROMB MATHS E1	LOOSE ART B7	EGGLESTN S.S. G1	GALE MUSIC M1	RYAN ENG H1
3WS	WILLIAMS S.S. P3	FLEMING H.EC. P3 GALE MUSIC M1	MOORE M.W. D7 ANDERSON W.W. D6	DRAVTZKI SCI H4	DRAVTZKI SCI H4	WILLIAMS ENG P3	THOMAS MATHS E6
4BN	BROWN-RC S.S. H3	FLEMING H.EC. P3 GALE MUSIC M1	MOORE M.W. D7 ANDERSON W.W. D6	LEARY P.E.	RICHARDS SCI C3	BROWN-RC ENG H3	BROWN-RC ENG H3
4LO	GALE MUSIC M1	HEINZ IDS F6	BURGESS FR. H7	WILKNSON S.S. G7	TOVEY ENG P5	THOMAS MATHS E6	WILKNSON S.S. G7
4BT	CROMB MATHS E7	HEMINGSN W.W. D6	EAGLE FR. P2	DUNCAN S.S. G4	INNES ENG F1	BENNETT ART B6	BENNETT ART B6
4BK	THOMAS MATHS E6	MOORE M.W. D7	INNES GER F1	TOVEY S.S. P5	BLACK SCI F5	LOOSE ART B7	LOOSE ART B7
4BR	LEAN ENG M2	SIEPKES H.EC. C6 FINNEGAN C.ST. C4	SKINNER C.ST. C4	EGGLESTN S.S. G1	BROWN-MD SCI F4	TRENBRTH MATHS F6	TRENBRTH MATHS F6
4PR	WILKNSON S.S. G7	CROSS SCI C1	CROSS SCI C1	FLEMING H.EC. C7	FLEMING H.EC. C7	GIBSON TYP C5	WILLIAMS ENG P3
4MCA	MCAULIFF MATHS F7	HARKNESS SCI C3	FAASS S.S. G6	ANDERSON W.W. D6	ANDERSON W.W. D6	ANDERSON T.D. D5	JOHNSON ENG B5
4CL	FAASS S.S. G6	LEARY P.E. MILLER P.E.	CLARK SCI E4	SHERRARD M.W. D7 SIEPKES CLO C6	SHERRARD M.W. D7 SIEPKES CLO C6	SKINNER C.ST. C4	CAMPBELL ENG F3
4SD	GIBSON C.ST. C4	" " " "	THURSTON S.S. G5	RYAN ENG H1	RYAN ENG H1	SIDAWAY SCI D4	BLACK MATHS F5
4CO	ANDERSON	EAGLE	MCSWENEY	BOOTH	MCNEILE	MCSWENEY	CONRADSN

Figure D.4 Teacher Master Timetable

MONDAY

	1	2	3	4	5	6	7
HITCHING				5WN SCI C3			
POOLE			F.6 HIST P3				
HEINZ		4LO-4BR IDS F6		F.6 ENG G6	F.6 ENG G6		
BENNETT		3EG ART B6	3EG ART B6	3SK ART B6	F.5 ART B6	4BT ART B6	4BT ART B6
BLACK	5RY-5CA PHX F5	3CS MATHS F5	F.6 PHX F5	3CS MATHS F5	4BK SCI F5	5TB SCI F4	4SD MATHS F5
BOOTH	3TY/3WI P.E.		F.5 P.E.	4CO/4WA P.E.	F.5 P.E.	3RS/3SK P.E.	3MCN/3EA P.E.
BROWN-MD	3EA SCI F4		F.6 PHX F4	7HN PHX F4	4BR SCI F4	7HN G.ST.	7HN G.ST.
BROWN-RC	4BN S.S. H3	5TB HIST H3	3TY S.S. H3	3MCN S.S. H3	3TY S.S. H3	4BN ENG H3	4BN ENG H3
BURGESS	3CS ENG H7		4LO-4BR FR. H7	3MCS-3WI FR. H7		F.6 FR. H7	F.6 FR. H7
CAMPBELL	5RY-5CA GER F3	3MCN GER F3		5TB ENG F3	3EG ENG F3		4CL ENG F3
CLARK	F.6 BIOL E4	3CR SCI E4	4CL SCI E4		7HN BIOL E4	4WA SCI E4	4WA SCI E4
CONRADSN	7HN HIST G3	5HS-5CA HIST G3	3EA ENG G3	F.6 ENG G3	F.6 ENG G3		4CO S.S. G3
CROMB	4BT MATHS E7		3CR MATHS E1	F.5 MATHS E7			
CROSS	5RY-5CA SCI C1	4PR SCI C1	4PR SCI C1		3CS SCI C1	F.6 CHEM C1	F.6 CHEM C1
DRAVTZKI	F.6 BIOL H4	5EL BIOL H4		3WS SCI H4	3WS SCI H4		5TB SCI H4
DUNCAN	5RY-5CA GEOG G4	6DN GEOG G4	3BE S.S. G4	4BT S.S. G4	3BE S.S. G4	F.6 S.S. G4	F.6 S.S. G4
EAGLE	5RY-5CA FR. P2	4CO ENG P2	4LO-4BR FR. P2	F.6 ENG P2	F.6 ENG P2		3CS-3BE FR. P2
EDIE	3BE MATHS E5	6ED MATHS E5	F.5 MATHS E5	3BE MATHS E5	F.5 MATHS E5		3SK MATHS E5
EGGLESTN	3EG S.S. G1	5RY HIST G1	F.6 HIST G1	4BR S.S. G1	3CR S.S. G1	3EG S.S. G1	
ELL	5EL ENG H6	4WA ENG H6	4WA ENG H6	F.6 ENG H6	F.6 ENG H6	5CA ENG H6	5CA ENG H6
FAASS	4CL S.S. G6	3RS S.S. G6	4MCA S.S. G6				
FLEMING	3RS-3CR H.EC. C7	3WS/4BN H.EC. P3	F.5 H.EC. C7	4PR-4CL H.EC. C7	4PR-4CL H.EC. C7	3MCS-3WI H.EC. C7	3MCS-3WI H.EC. C7
GALE	4LO MUSIC M1	3WS/4BN MUSIC M1	3CS MUSIC M1	3EG MUSIC M1		3CR MUSIC M1	3RS MUSIC M1

Figure D.5 Teacher Individual Timetables

BENNETT

	1	2	3	4	5	6	7
MON		3EG ART B6	3EG ART B6	3SK ART B6	F.5 ART B6	4BT ART B6	4BT ART B6
TUE	3BE ART B6	3BE ART B6	F.5 ART B6	F.5 ART B6	3WI ART B6	F.6 ART B6	F.6 ART B6
WED	4BR ART B6	4BR ART B6		4SD ART B6	4SD ART B6	4CL ART B6	4CL ART B6
THU	4MCA ART B6	4MCA ART B6	F.5 ART B6	F.5 ART B6	3EA ART B6	3EA ART B6	4WA ART B6
FRI	3MCS ART B6	3MCS ART B6	3TY ART B6	3TY ART B6	F.5 ART B6	F.6 ART B6	

BLACK

	1	2	3	4	5	6	7
MON	5RY-5CA PHX F5	3CS MATHS F5	F.6 PHX W	3CS MATHS F5	4BK SCI F5	5TB SCI F4	4SD MATHS F5
TUE	4SD MATHS F5	3CS MATHS F5	4BK SCI F4		5RY-5CA PHX F5	F.6 PHX F4	F.6 PHX F4
WED	5RY-5CA PHX F4	5TB SCI F4	4BK SCI C1		3CS MATHS F5	F.6 PHX F5	F.6 PHX F5
THU	4SD MATHS F5	5TB SCI F4	4BK SCI F4	4BK SCI F4	5RY-5CA PHX F4	5RY-5CA PHX F4	4SD MATHS F5
FRI	3CS MATHS F5		5RY-5CA PHX F4	4SD MATHS F5	4BK SCI F5	F.6 PHX F4	5RY-5CA PHX F5

Figure D.6 Classroom Master Timetable

MONDAY

	1	2	3	4	5	6	7
B5	FINNEGAN 5RY-5CA		JOHNSON 7HN	JOHNSON F.6	JOHNSON F.6	JOHNSON 5JN	JOHNSON 4MCA
B6	SKINNER 5RY-5CA	BENNETT 3EG	BENNETT 3EG	BENNETT 3SK	BENNETT F.5	BENNETT 4BT	BENNETT 4BT
B7		LOOSE R/O	LOOSE F.6	LOOSE 3CR	LOOSE F.5	LOOW 4BK	LOOSE 4BK
C1	CROSS 5RY-5CA	CROSS 4PR	CROSS 4PR	HARKNESS 5EL	CROSS 3CS	CROSS F.6	CROSS F.6
C3	RICHARDS 7HN	HARKNESS 4MCA	RICHARDS 3RS	HITCHING 5WN	RICHARDS 4BN	RICHARDS F.6	RICHARDS F.6
C4	GIBSON 4SD-4WA	FINNEGAN 4LO-4BR	SKINNER 4LO-4BR	FINNEGAN 3MCS-3WI		SKINNER 4P4CL	SKINNER 3CS-3BE
C5	WINN 4SD-4WA		WINN F.5	SKINNER 3MCS-3WI	WINN F.5	GIBSON 4PR-4CL	WINN 3CS-3BE
C6	SIEPKES 3RS-3CR	SIEPKES 4LO-4BR	SIEPKES F.5	SIEPKES 4PR-4CL	SIEPKES 4PR-4CL	SIEPKES 3MCS-3WI	SIEPKES 3MCS-3WI
C7	FLEMING 3RS-3CR		FLEMING F.5	FLEMING 4PR-4CL	FLEMING 4PR-4CL	FLEMING 3MCS-3WI	FLEMING 3MCS-3WI
D1	MCSWENEY 5RY-5CA	SIDAWAY 3SK	SIDAWAY 3SK	SIDAWAY 5RY	SIDAWAY 3WI	MILLER 5FS	MILLER 5FS
D3	SHERRARD F.6	SHERRARD 5HS-5CA	SHERRARD F.5	MOORE 3MCS-3WI		MOORE 5AN	
D4	SIDAWAY F.6	MCSWENEY 3TY	MCSWENEY 4CO		MCSWENEY 3MCS	SIDAWAY 4SD	MCSWENEY 5AN
D5	ANDERSON 4SD-4WA	ANDERSON 5HS-5CA			HEMINGSN F.5	ANDERSON 4PR-4CL	ANDERSON 3CS-3BE
D6	HEMINGSN 3RS-3CR	HEMINGSN 4LO-4BR	ANDERSON 3WS/4BN	ANDERSON 4PR-4CL	ANDERSON 4PR-4CL	HEMINGSN 3MCS-3WI	HEMINGSN 3MCS-3WI
D7	MOORE 3RS-3CR	MOORE 4LO-4BR	MOORE 3WS/4BN	SHERRARD 4PR-4CL	SHERRARD 4PR-4CL	SHERRARD 3MCS-3WI	SHERRARD 3MCS-3WI
E1	MCNEILE 3MCN	MCNEILE 3BE/3MCS	CROMB 3CR	MCNEILE F.5	MCNEILE 4CO	MCNEILE 3MCN	
E3	PORTER 6GN	PORTER 3EA	PORTER F.5	PORTER 7HN	PORTER F.5	PORTER F.6	PORTER F.6
E4	CLARK F.6	CLARK 3CR	CLARK 4CL	MILLER 5FS	CLARK 7HN	CLARK 4WA	CLARK 4WA
E5	EDIE 3BE	EDIE 6ED	EDIE F.5	EDIE 3BE	EDIE F.5		EDIE 3SK
E6	THOMAS 4BK		THOMAS F.5	THOMAS 3RS	THOMAS F.5	THOMAS 4LO	THOMAS 3WS
E7	CROMB 4BT	HENDERSN 6HD		CROMB F.5	HENDERSN 4WA	HENDERSN 5IN	HENDERSN 5IN
F1	INNES F.6	INNES 3WI	INNES 4LO-4BR	INNES 5IN	INNES 4BT	INNES 3EA	
F3	CAMPBELL 5RY-5CA	CAMPBELL 3MCN		CAMPBELL 5TB	CAMPBELL 3EG		CAMPBELL 4CL
F4	BROWN-MD 3EA		BROWN-MD F.6	BROWN-MD 7HN	BROWN-MD 4BR	BLACK 5TB	SIDAWAY 5RY
F5	BLACK 5RY-5CA	BLACK 3CS	BLACK F.6	BLACK 3CS	BLACK 4BK		BLACK 4SD
F6	TRENBRTH 5TB	HEINZ 4LO-4BR		TRENBRTH 5AN	TRENBRTH 7HN	TRENBRTH 4BR	TRENBRTH 4BR
F7	MCAULIFF 4MCA	MCAULIFF 7HN	MCAULIFF 3WI	MCAULIFF F.5		MCAULIFF F.6	MCAULIFF F.6

Figure D.7 Condensed Class Master Timetable

MONDAY

	1	2	3	4	5	6	7
3MCN	MNMTH E1	CAGER F3	MGENG P1	BNS.S H3	HSSCI H5	MNMTH E1	BTP.E
3EA	BRSCI F4	PRMTH E3	COENG G3	TNS.S G5	WSENG P3	INGER F1	WAP.E
3CS	BUENG H7	BKMTH F5	GLMUS M1	BKMTH F5	CSSCI C1	WIS.S G7	EAFR. P2
3EG	EGS.S G1	BEART B6	BEART B6	GLMUS M1	CAENG F3	EGS.S G1	ANT.D D5
3BE	EDMTH E5	MNP.E E1	DNS.S G4	EDMTH E5	DNS.S G4	WNENG P2	SKCST C4
		WAP.E					WNTYP C5
3MCS	RYENG H1	" "	LTS.S H1	BUFR. H7	MSSCI D4	FLHEC C7	FLHEC C7
3TY	BTP.E	MSSCI D4	BNS.S H3	MOT.D D3	BNS.S H3	HNW.W D6	HNW.W D6
3WI	WAP.E	INENG F1	MAMTH F7	SKTYP C5	SDSCI D1	SHM.W D7	SHM.W D7
				FGCST C4		SPCLO C6	SPCLO C6
3RS	FLHEC C7	FSS.S G6	RSSCI C3	THMTH E6	LEENG M1	BTP.E	GLMUS M1
3SK	HNW.W D6	SDSCI D1	SDSCI D1	BEART B6	TNS.S G5	WAP.E	EDMTH E5
3CR	MOM.W D7	CLSCI E4	CRMTH E1	LOART B7	EGS.S G1	GLMUS M1	RYENG H1
	SPCLO C6						
3WS	WSS.S P3	FLHEC P3	MOM.W D7	DASCI H4	DASCI H4	WSENG P3	THMTH E6
		GLMUS M1	ANW.W D6				
4BN	BNS.S H3	FLHEC P3	MOM.W D7	LYP.E	RSSCI C3	BNENG H3	BNENG H3
		GLMUS M1	ANW.W D6				
4LO	GLMUS M1	HZIDS F6	BUFR. H7	WIS.S G7	TYENG P5	THMTH E6	WIS.S G7
4BT	CRMTH E7	HNW.W D6	EAFR. P2	DNS.S G4	INENG F1	BEART B6	BEART B6
4BK	THMTH E6	MOM.W D7	INGER F1	TYS.S P5	BKSCI F5	LOART B7	LOART B7
4BR	LEENG M2	SPHEC C6	SKCST C4	EGS.S G1	BRSCI F4	TBMTH F6	TBMTH F6
		FGCST C4					
4PR	WIS.S G7	CSSCI C1	CSSCI C1	FLHEC C7	FLHEC C7	GNTYP C5	WSENG P3
4MCA	MAMTH F7	HSSCI C3	FSS.S G6	ANW.W D6	ANW.W D6	ANT.D D5	JNENG B5
4CL	FSS.S G6	LYP.E	CLSCI E4	SHM.W D7	SHM.W D7	SKCST C4	CAENG F3
		MLP.E		SPCLO C6	SPCLO C6		
4SD	GNCST C4	" "	TNS.S G5	RYENG H1	RYENG H1	SDSCI D4	BKMTH F5
4CO	ANT.D D5	EAENG P2	MSSCI D4	BTP.E	MNMTH E1	MSSCI H5	COS.S G3
4WA	WNTYP C5	ELENG H6	ELENG H6	WAP.E	HDMTH E7	CLSCI E4	CLSCI E4
	MMMST P1						
5TB	TBMTH F6	BNHIS H3	GNTYP M2	CAENG F3	GNTYP M2	BKSCI F4	DASCI H4
5EL	ELENG H6	DABIO H4	WIGEO G7	HSSCI C1	WIGEO G7	TNGEO G5	TNGEO G5
			WNTYP C5		WNTYP C5		
5RY	BKPHX F5	EGHIS G1	EDMTH E5	SDSCI D1	EDMTH E5	RYENG H1	SDSCI F4
5IN	CAGER F3	MGGEO P1	PRMTH E3	INENG F1	PRMTH E3	HDMTH E7	HDMTH E7
5HS	CSSCI C1	COHIS G3	THMTH E6	WSENG P3	THMTH E6	HSBIO H4	HSBIO H5
5FS	DNGEO G4	ANT.D D5	FLHEC C7	MLBIO E4	BTP.E	MLBIO D1	MLBIO D1
5WN	EAFR. P2	RYGEO H1	SHM.W D3	HMSCI C3	MLP.E	MGENG P1	MGENG P1
5LE	MSSCI D1	SHT.D D3	SPCLO C6	CRMTH E7	" "	LEENG M2	LEENG M2
5JN	SKCPR B6	LTGEO H7	BTP.E	MAMTH F7	BEART B6	JNENG B5	TYENG P5
5CA	TNGEO G5	" "	WAP.E	MNMTH E1	HNW.W D5	ELENG H6	ELENG H6
	FGCPR B5		MLP.E		LOART B7		
5AN	TYENG P5	TYS.S P5	" "	TBMTH F6	" "	MOT.D D3	MSSCI D4
6ED	CLBIO E4	EDMTH E5	DPHIS P3	HZENG G6	HZENG G6	BUFR. H7	BUFR. H7
6HD	DABIO H4	HDMTH E7	BKPHX F5	COENG G3	COENG G3	CSCHM C1	CSCHM C1
6DN	HSBIO H5	DNGEO G4	BRPHX F4	EAENG P2	EAENG P2	DNS.S G4	DNS.S G4
6TN	INGER F1	TNGEO G5	EGHIS G1	ELENG H6	ELENG H6	MAMTH F7	MAMTH F7
	SHT.D D3		LYP.E H5	JNENG B5	JNENG B5	PRMTH E3	PRMTH E3
	SDBIO D4		LOART B7			RSCHM C3	RSCHM C3
6GN	PRCPS E3	GNSEC M2	" "	" "	" "	FGACC G6	FGACC G6
7HN	COHIS G3	MAMTH F7	JNENG B5	BRPHX F4	CLBIO E4	BRGST	BRGST
	RSCHM C3	WIGEO G7	LEENG P5	PRMTH E3	TBADM F6		
				WNACC P1			